版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

Broadview
www.broadview.com.cn



微信公众平台
企业应用开发实战

刘捷◎编著

资深微信公众平台企业应用开发专家力作

详细解读微信公众平台应用开发所需的各种技术、方法和技巧
深入剖析微信公众平台企业级应用开发系统架构及开发过程
轻松面对世界500强级别复杂业务逻辑企业应用开发

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

刘捷

上海谦讯网络科技有限公司CTO，国内微信公众平台应用开发先驱。于2013年年初进入微信公众平台应用开发领域，专注于微信公众平台企业应用开发与实施，已为10多个世界500强客户建立起完整的微信公众号企业管理平台，带领团队开发过数十个微信公众平台企业级应用，涉及银行、保险、医药、地产、汽车等众多行业，具有丰富的微信公众平台项目开发与实施经验。



微信公众平台
企业应用开发实战

刘捷◎编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书全面介绍了微信公众平台已开放 API 的功能、使用方法及应用场景，详细解读了微信公众平台应用开发所需的各种技术、方法和技巧，深入讲解了微信公众平台开源开发框架 Senparc.Weixin.MP。书中还精选了几个具有代表性的商用开发实际案例，以 C#开发语言为例，系统讲解了微信公众平台企业应用开发的系统架构及完整的开发过程。

全书共 8 章，可分为四个部分：第一部分（第 1 章）介绍了微信公众平台的基础知识，为读者学习后续章节打下基础；第二部分（第 2、3 章）介绍了进行微信公众平台开发所需的准备工作及将应用接入微信公众平台的方法；第三部分（第 4、5 章）详细介绍了微信公众平台已开放的 API，通过实战项目对开发框架进行了系统讲解，读者甚至可以直接使用开发框架进行自己的应用开发；第四部分（第 6~8 章）向读者展示了 3 个企业应用的实际开发过程。通过本书的学习，读者将完全有能力胜任大型微信公众平台企业应用开发工作。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

微信公众平台企业应用开发实战 / 刘捷编著. —北京：电子工业出版社，2015.1
ISBN 978-7-121-25013-2

Ⅰ. ①微… Ⅱ. ①刘… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字(2014)第 280484 号

策划编辑：牛　勇
责任编辑：徐津平
文字编辑：杨　璐
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：720×1000　1/16　印张：20.5　字数：394 千字
版　　次：2015 年 1 月第 1 版
印　　次：2015 年 6 月第 3 次印刷
定　　价：49.80 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

作为企业与超过 6 亿微信用户接触的主要渠道，微信公众平台在过去的一年中有了巨大的发展。截至 2014 年 7 月，微信公众账号的数量已超过 580 万个，日均增长数由 2013 年的 8000 个上升至 1.5 万个。

微信公众平台只提供了一些基础的功能，如群发消息、消息回复等，但这些基础功能并不能满足许多企业为其客户在微信上提供服务的需求。为此，微信公众平台开放了一系列的开发接口，企业可通过调用这些接口，开发微信公众平台应用并接入微信公众平台，实现在微信中开展现有业务或提供客户服务。

随着微信公众平台的发展，微信公众平台应用开发的需求越来越多。笔者在 2013 年年初进入微信公众平台应用开发领域，在一年多的时间里，团队开发了很多微信公众平台企业应用。在为各企业客户进行应用开发的过程中，实现了许多不同的需求，也碰到了不少困难，积累了一些开发经验，也对微信公众平台有了比较深入的了解。

本书就是将这些开发经验和对微信公众平台的理解进行系统、全面地整理，然后分享给读者，希望读者能够从中得到一些帮助，也希望借此机会认识更多的同行。

读者对象

对微信公众平台感兴趣，准备使用或已使用微信公众平台的朋友。

从事微信公众账号的运营、推广、管理等工作的从业人员。

对微信公众平台企业应用有需求的IT人员或市场部人员。

准备从事或已从事微信公众平台应用开发的工程师。

进行微信公众平台企业应用开发的企业内部人员或第三方开发团队。

如何阅读本书

本书的内容在逻辑上可分为四个部分。

第一部分（第1章）介绍了微信公众平台的基础知识，包含微信公众账号的类型及各类型的特点和限制、微信公众账号的注册流程、微信公众账号的认证流程及最新增加的功能插件介绍，为读者学习后续章节打下基础。

第二部分（第2、3章）介绍了进行微信公众平台开发所需的准备工作，以及将应用接入微信公众平台的方法、操作步骤和示例程序。

第三部分（第4、5章）详细介绍了微信公众平台已开放的API，包括基础的消息、事件接口、自定义菜单接口及认证服务号拥有的9个高级接口；对笔者团队在实际项目中使用的进行完善和重构后的微信公众平台开源开发框架Senparc.Weixin.MP进行了系统讲解；全面剖析了所有的源代码，并使用开发框架完成了一个能够接收与响应任何类型消息的项目。读者学习完本部分后，可以建立自己的开发框架，也可以直接使用开发框架进行应用开发。以开发框架作为基础，在开发微信公众平台应用时，只需要关注业务逻辑。

第四部分（第6~8章）依次向读者展示了预约、阅读统计、渠道管理等3个实际的企业应用的开发过程。学习并掌握本部分知识后，读者可以独立完成具有复杂业务逻辑的微信公众平台应用开发，并完全有能力胜任大型微信公众平台企业应用开发工作。

如果读者不懂程序开发，想了解微信公众平台，那么请重点阅读第一部分；如果读者刚接触微信公众平台，那么请务必从第一部分的基础知识开始学习；如果读者有微信公众平台应用开发的经验，那么可以选择自己感兴趣的章节阅读。

图书下载资源请参见 http://www.broadview.com.cn/25013

勘误和支持

因笔者的水平有限，虽然已尽力来完善此书，书中仍难免会出现一些错误、不准确或考虑不周的地方，恳请读者批评、指正。

致谢

感谢微信团队，是他们创造了这款伟大的移动社交产品。

感谢微信公众平台开源开发框架 Senparc.Weixin.MP 的项目开发团队，依靠他们建立的优秀开发框架，才使我们可以专注于业务逻辑的开发。

感谢参加本书编写工作的其他成员：曹洪匪、李彪、邓建功、胥桂蓉、唐蓉、朱世波、尹新梅、李勇、杨任毅、王政、黄刚、赵阳春、何紧莲、邓春华。尤其感谢成都鼎翰文化的邓建功老师和电子工业出版社的牛勇老师，感谢你们在我写作过程中提供的支持，正是你们的鼓励和帮助，才使我能顺利完成全部书稿。

谨以此书献给我最亲爱的家人！

刘　捷

2014 年 10 月

目　录

第 1 章　微信公众平台入门 1

1.1　微信公众平台简介 2

1.2　微信公众号类型 2

1.2.1　订阅号特点 2

1.2.2　服务号特点 3

1.2.3　微信公众号不同类型的区别 3

1.3　微信公众号注册 5

1.3.1　注册微信公众号基本信息 5

1.3.2　邮箱激活微信公众号 6

1.3.3　登记微信公众号信息 7

1.3.4　选择微信公众号类型 11

1.3.5　填写微信公众号信息 12

1.4　微信认证 13

1.4.1　微信认证优势 14

1.4.2　微信认证流程 15

1.4.3　微信认证结果 18

1.5　功能插件简介 19

1.5.1　多客服 19

1.5.2 微信支付 23
1.5.3 微信小店 27

第 2 章 微信公众平台开发准备 35

2.1 成为微信公众平台开发者 35
2.2 微信公众平台接口测试账号申请 38

第 3 章 实现 URL 接入 43

3.1 接口校验方法 43
3.2 实现接口校验程序 45
3.3 本地测试 48
3.4 使用 AppHarbor 的部署接口校验程序 49
3.5 接入微信公众平台 56

第 4 章 微信公众平台消息处理框架 57

4.1 消息交互基础 57
4.1.1 消息交互流程 58
4.1.2 消息数据结构 58
4.1.3 用户发送消息数据实体 60
4.1.4 用户发送事件消息数据实体 64
4.1.5 公众号回复消息数据实体 68
4.1.6 消息数据转换 73
4.2 用户会话上下文框架 85
4.2.1 用户会话上下文应用场景 85
4.2.2 用户会话上下文结构 86
4.2.3 发送与接收消息记录 86
4.2.4 用户会话上下文信息 88
4.2.5 用户会话上下文集合 91
4.3 消息处理 98

4.3.1 消息处理完整流程 98
4.3.2 实现消息处理 100
4.4 消息处理框架的完整结构 112
4.5 消息处理框架使用示例 113
4.5.1 消息处理框架示例程序 114
4.5.2 在 AppHarbor 部署示例程序 125
4.5.3 示例程序运行结果 126

第 5 章 微信公众平台接口开发框架 129

5.1 微信公众平台接口基础 129
5.1.1 高级接口交互流程 130
5.1.2 实现 HTTPS 请求 131
5.1.3 封装接口访问方法 133
5.2 获取接口访问凭证 138
5.3 自定义菜单接口 143
5.3.1 自定义菜单简介 143
5.3.2 自定义菜单数据结构 144
5.3.3 自定义菜单数据实体 145
5.3.4 自定义菜单接口封装 149
5.4 多媒体文件接口 155
5.4.1 多媒体文件接口简介 155
5.4.2 上传下载文件 157
5.4.3 多媒体文件接口封装 160
5.5 用户管理接口 163
5.5.1 用户信息接口简介 164
5.5.2 用户信息接口封装 166
5.5.3 用户分组接口简介 169
5.5.4 用户分组接口封装 172
5.6 客服接口 176

5.6.1 客服接口简介 176
5.6.2 客服接口封装 181
5.7 生成带参数的二维码接口 185
5.7.1 带参数二维码接口简介 185
5.7.2 带参数二维码接口封装 187
5.8 网页授权接口 189
5.8.1 网页授权接口简介 189
5.8.2 网页授权接口封装 194

第 6 章 商用案例 1——预约系统 200

6.1 预约系统需求 200
6.2 预约系统功能及设计 201
6.2.1 预约系统功能 201
6.2.2 不定字段数目的数据库表和数据结构设计 202
6.2.3 数据表设计 204
6.3 预约系统架构实现 206
6.3.1 商用系统三层架构简述 206
6.3.2 预约系统三层架构搭建 208
6.3.3 实现数据访问框架 208
6.3.4 实现数据访问层 210
6.3.5 实现视图实体层 218
6.3.6 实现业务逻辑层 223
6.4 预约系统实现 226
6.4.1 预约系统后台实现 226
6.4.2 预约系统前台实现 231
6.5 部署及测试体验 234

第 7 章 商用案例 2——阅读、分享统计 236

7.1 阅读、分享统计的意义 236

7.2 获取分享记录 237
7.2.1 微信 JS 接口简介 237
7.2.2 使用微信 JS 接口获取分享记录 238
7.3 获取访问来源 241
7.4 识别访问者与分享者 243
7.4.1 识别访问者 243
7.4.2 识别分享者 244
7.4.3 实现识别访问者与分享者 245
7.5 阅读、分享统计实现 250
7.5.1 内存数据库实现数据存取 250
7.5.2 实现阅读、分享数据记录 254
7.5.3 实现阅读、分享统计 260
7.6 部署及测试体验 269

第 8 章 商用案例 3——推广渠道管理系统 271

8.1 微信公众号推广综述 271
8.2 推广渠道管理系统功能及设计 273
8.2.1 推广渠道管理系统需求 273
8.2.2 推广渠道管理系统功能 274
8.2.3 数据表设计 275
8.3 推广渠道管理系统实现 277
8.3.1 实现数据访问层 278
8.3.2 实现视图实体层 281
8.3.3 同步微信个人用户信息 288
8.3.4 实现业务逻辑层 294
8.3.5 推广渠道管理系统后台实现 301
8.4 部署及测试体验 318

第 1 章
微信公众平台入门

微信是腾讯公司于 2011 年 1 月 21 日推出的一个为智能终端提供即时通信服务的免费应用程序，截至 2013 年 11 月注册用户量已经突破 6 亿，是亚洲地区最大用户群体的移动即时通信软件。微信已成为移动互联网最大的入口之一。

微信公众平台是微信生态圈的重要组成部分，作为企业与微信 6 亿多用户联系的纽带，截至 2013 年 11 月，已经有 200 多万的企业、组织与个人注册了微信公众平台账号，并且每天有 8000 左右的新用户注册微信公众平台。

本书的重点就是帮助读者了解微信公众平台，熟悉微信公众平台的注册流程与如何使用微信公众平台的基本功能，掌握微信公众平台开发的相关知识与技术，从而能更好地使用微信公众平台及开展微信公众平台的开发工作。

本章将为读者介绍微信公众平台的基础知识。

1.1 微信公众平台简介

微信公众平台是腾讯公司在微信的基础上新增的功能模块，通过这一平台，个人和企业都可以打造一个微信公众号，并实现和特定群体的文字、图片、语音的全方位沟通、互动。目前微信公众平台支持 PC 端网页、移动互联网客户端登录，并可以绑定私人账号进行信息群发。

微信公众号主要面向名人、政府、媒体、企业等机构推出的合作推广及用户服务业务。在这里可以通过微信渠道将品牌推广给 6 亿的微信用户，减少宣传成本，提高品牌知名度，提升服务体验，打造更具影响力的品牌形象。

1.2 微信公众号类型

微信公众号分为订阅号与服务号等两种类型。服务号能给企业和组织提供更强大的业务服务与用户管理能力，帮助企业快速实现全新的公众号服务平台。订阅号能为媒体和个人提供一种新的信息传播方式，构建与读者之间更好的沟通与管理模式。

为了确保公众账号信息的真实性、安全性，微信公众平台提供给订阅号与服务号进行微信认证的服务。通过微信认证后，微信公众号将获得更丰富的高级接口，以向用户提供更有价值的个性化服务。同时，通过微信认证后，用户将在微信公众号的详细信息中看到微信认证特有的标识，并在认证详情中查看微信公众号的运营主体信息。

1.2.1 订阅号特点

订阅号主要为用户提供信息和资讯，一般媒体用户比较适合，如骑行西藏、央视新闻等。订阅号有以下几个特点：

◎ 每天可以发送 1 条群发消息（每天 0 点更新，次数不会累加），群发消息收至订阅号文件夹，群发消息不会提示推送。

◎ 发给订阅用户（粉丝）的消息，将会显示在对方的“订阅号”文件夹中。

◎ 在订阅用户（粉丝）的通讯录中，订阅号将被放入订阅号文件夹中。

◎ 订阅号具有关键词回复等基础接口。

◎ 订阅号认证后可申请自定义菜单。

1.2.2　服务号特点

服务号主要为用户提供服务。一般银行和企业用户比较适合，如招商银行、中国南方航空等。服务号特点如下：

◎ 1 个月（30 天）内仅可以发送 4 条群发消息，群发消息显示在聊天列表，粉丝会收到新消息提醒。

◎ 发给订阅用户（粉丝）的消息，会显示在对方的聊天列表中。

◎ 服务号会在订阅用户（粉丝）的通讯录中。

◎ 服务号可申请自定义菜单。

◎ 服务号具有关键词回复等基础接口。

◎ 服务号认证后可开通 10 大高级接口。

◎ 服务号认证后可使用多客服功能。

◎ 服务号认证后可申请微信支付功能。

◎ 服务号认证后可申请微信小店功能。

1.2.3　微信公众号不同类型的区别

订阅号每天能群发 1 条消息，但所有的订阅号都在订阅号文件夹中，微信个人用户不会收到推送提示。服务号每月只能群发 4 条，且群发的消息会显示在聊天列表中，微信个人用户会收到推送提示。

在功能上，订阅号经过微信认证也只是增加了自定义菜单接口，对认证服务号开放的高级接口和高级功能，订阅号都不能使用。

各类型微信公众号的详细区别见表 1-1。

表 1-1

功　能	未认证订阅号	认证订阅号	未认证服务号	认证服务号
微信显示位置	订阅号文件夹	订阅号文件夹	聊天列表	聊天列表
群发消息	每天 1 条	每天 1 条	每月 4 条	每月 4 条
自动回复	有	有	有	有
自定义菜单	无	有	有	有
微信支付	无	无	无	有
多客服	无	无	无	有
微信小店	无	无	无	有
消息管理	有	有	有	有
用户管理	有	有	有	有
素材管理	有	有	有	有
统计	有	有	有	有
开发者中心	有	有	有	有
基础接口	有	有	有	有
高级接口	无	无	无	有

其中，基础接口包括：

◎ 接收用户消息

◎ 向用户回复消息

◎ 接受事件推送

高级接口包括：

◎ 语音识别

◎ 客服接口

◎ 高级群发接口

◎ 修改 OAuth 2.0 网页授权

◎ 生成带参数二维码

◎ 开启获取用户地理位置

◎ 获取用户基本信息

◎ 获取关注者列表

◎ 用户分组接口

◎ 上传下载多媒体文件

拥有自定义菜单功能的微信公众号，将自动获得自定义菜单接口权限。

1.3　微信公众号注册

通过前面的介绍，相信读者对微信公众平台和微信公众号已经有了一个大概的认识，下面为读者介绍如何注册一个微信公众号。

1.3.1　注册微信公众号基本信息

首先，通过计算机登录微信公众平台官网，网页地址为 http://mp.weixin.qq.com/，单击右上角的“立即注册”，如图 1-1 所示。

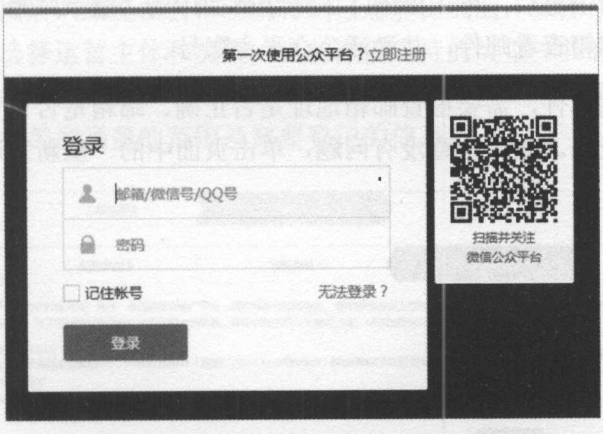


图 1-1

在新打开的注册页面填写注册邮箱，设置公众号登录密码，填写完验证码后单击“注册”按钮，微信团队会向我们填写的注册邮箱发送一封账号激活邮件，如图 1-2 所示。

一个邮箱只能注册一个微信公众号，请使用未绑定微信的邮箱进行注册。

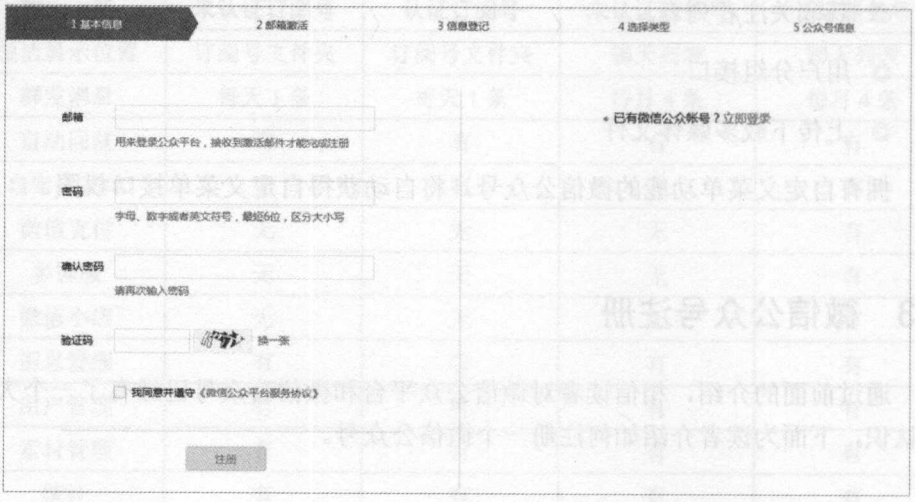


图 1-2

1.3.2 邮箱激活微信公众号

提交完注册信息后，单击如图 1-3 所示界面中的“登录邮箱”按钮，登录到注册时填写的邮箱查看邮件，并激活公众平台账号。

若没有收到邮件，需要检查邮箱地址是否正确，邮箱是否设置了邮件过滤或查看邮件的垃圾箱。如果检查没有问题，单击页面中的“重新发送”按钮，尝试重新发送。

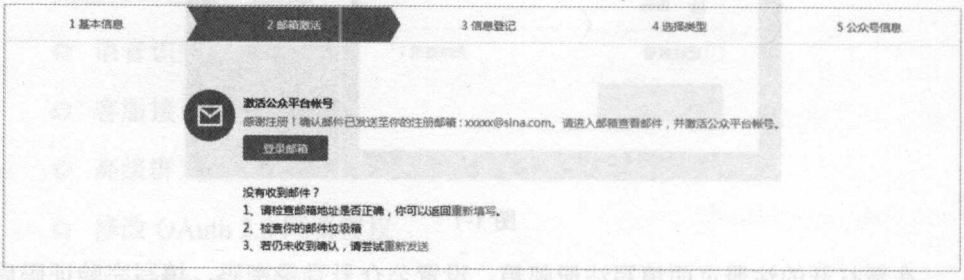


图 1-3

收到的激活邮件如图 1-4 所示，单击邮件中的链接地址，完成激活。激活邮

件中的链接地址只有 48 小时的有效期，超过 48 小时，需要重新注册，重新获取激活邮件。

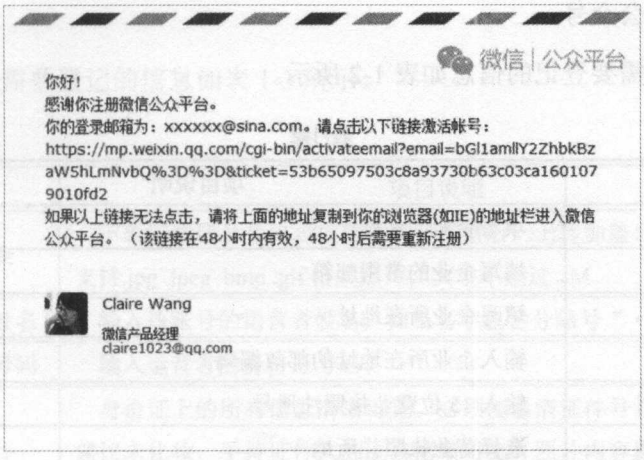


图 1-4

1.3.3　登记微信公众号信息

单击激活链接后，浏览器会自动跳转到信息登记页面，如图 1-5 所示。在信息登记页面需要选择运营主体和类型，目前微信支持的类型一共有 5 种，分别为政府、媒体、企业、其他组织和个人。不同的类型需要登记的信息不一样，下面我们逐一介绍 5 种类型涵盖的范围及需要登记的信息。

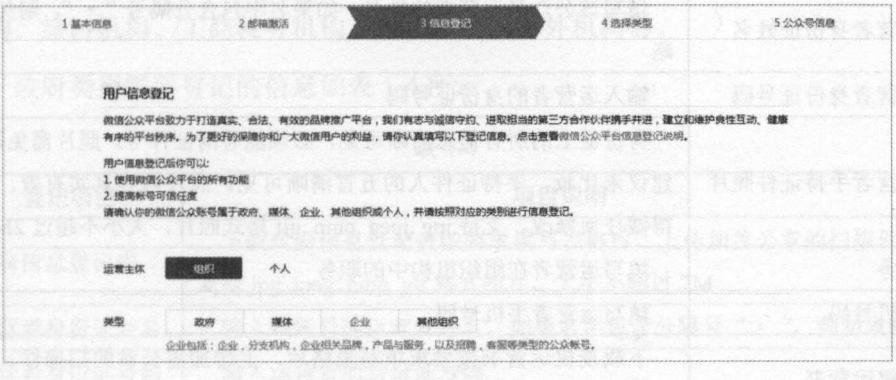


图 1-5

1. 企业类型

企业类型包括：企业、分支机构、企业相关品牌、产品与服务，以及招聘、客服等类型的公众号。

企业类型需要登记的信息如表 1-2 所示。

表 1-2

登记项目	项目说明
企业名称	填写企业的名称
企业邮箱	填写企业的常用邮箱
企业地址	填写企业所在地址
邮编	输入企业所在地址的邮政编码
营业执照注册号	输入 15 位营业执照注册号
营业执照住所地	选择营业执照住所地
成立日期	输入企业成立日期，如 2000 年 1 月 1 日
营业期限	填写企业的营业期限，如 10 年，若为长期则直接勾选
经营范围	填写企业的经营范围，如食品类
营业执照副本扫描件	上传营业执照清晰彩色原件扫描件或数码照，在有效期内且年检章齐全（当年成立的可无年检章）的情况下，由中国大陆工商局或市场监督管理局颁发。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
注册资本	填写企业的注册资本，单位：万元。如 10
组织机构代码	输入 9 位组织机构代码，如 12345678-9
运营者身份证姓名	填写该公众号运营者的姓名，如果名字包含分隔号“ · ”，则勿省略
运营者身份证号码	输入运营者的身份证号码
运营者手持证件照片	身份证上的所有信息清晰可见，必须能看清证件号。照片需免冠，建议未化妆，手持证件人的五官清晰可见。照片内容真实有效，不得做任何修改。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
职务	填写运营者在组织机构中的职务
手机号码	填写运营者手机号码
授权运营书	下载授权运营书按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M

2. 媒体类型

媒体类型包括：报纸、杂志、电视、电台、通信社、其他媒体等类型的公众号。

媒体类型需要登记的信息如表 1-3 所示。

表 1-3

登记项目	项目说明
媒体信息登记表	下载媒体信息登记表按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
运营者身份证姓名	输入该账号的运营者姓名，如果名字包含分隔号“ • ”，则勿省略
运营者身份证号码	输入运营者的身份证号码
运营者证件照片	身份证上的所有信息清晰可见，必须能看清证件号。照片需免冠，建议未化妆，手持证件人的五官清晰可见。照片内容真实有效，不得做任何修改。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
手机号码	输入运营者的手机号码
授权运营书	下载授权运营书按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M

3. 政府类型

政府类型包括：国内外、各级、各类政府机构、事业单位、具有行政职能的社会组织等类型的公众号，目前主要覆盖公安机构、党团机构、司法机构、交通机构、旅游机构、工商税务机构、市政机构、涉外机构等。

政府类型需要登记的信息如表 1-4 所示。

表 1-4

登记项目	项目说明
政府信息登记表	下载政府信息登记表按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
运营者身份证姓名	输入该账号的运营者姓名，如果名字包含分隔号“ • ”，则勿省略
运营者身份证号码	输入运营者的身份证号码

续表

登记项目	项目说明
运营者证件照片	身份证上的所有信息清晰可见，必须能看清证件号。照片需免冠，建议未化妆，手持证件人的五官清晰可见。照片内容真实有效，不得做任何修改。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
手机号码	输入运营者的手机号码
授权运营书	下载授权运营书按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M

4. 其他组织类型

其他组织类型包括：不属于企业、政府、媒体、个人的机构类型的公众号。

其他组织类型需要登记的信息如表 1-5 所示。

表 1-5

登记项目	项目说明
组织名称	填写该组织的名称
组织邮箱	填写该组织的常用邮箱
邮编	填写该组织所在地址的邮政编码
组织地址	填写该组织的所在地
组织电话	填写该组织的固定电话：区号-电话号码
组织机构代码	输入 9 位组织机构代码，如 12345678-9
组织机构代码扫描件	上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
运营者身份证姓名	填写该公众账号运营者姓名，如果名字包含分隔号“ • ”，则勿省略
运营者身份证号码	输入运营者的身份证号码
职务	填写运营者在组织机构中的职务
运营者手持证件照片	身份证上的所有信息清晰可见，必须能看清证件号。照片需免冠，建议未化妆，手持证件人的五官清晰可见。照片内容真实有效，不得做任何修改。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
手机号码	输入运营者的手机号码
授权运营书	下载授权运营书按要求填写表格后，上传加盖公章的扫描件。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M

5. 个人类型

个人类型包括：不属于企业、政府、媒体、其他组织类型的公众号。

个人类型需要登记的信息如表 1-6 所示。

表 1-6

登记项目	项目说明
身份证姓名	填写申请人的姓名，如果名字包含分隔号“ · ”，则勿省略
身份证号码	输入申请人的身份证号码
证件照片	身份证上的所有信息清晰可见，必须能看清证件号。照片需免冠，建议未化妆，手持证件人的五官清晰可见。照片内容真实有效，不得做任何修改。支持.jpg .jpeg .bmp .gif 格式照片，大小不超过 2M
手机号码	输入申请人的手机号码
城市	选择申请人所在地
固定电话	输入申请人的固定电话号码：区号-电话号码
单位名称	填写申请人所在单位的名称
职务	填写申请人当前的职位
个人住址(可选)	填写申请人当前的个人住址，可选填
单位地址(可选)	填写申请人所在单位的地址，可选填

根据申请的公众号类型登记完信息后，单击页面底部的“继续”按钮，如果信息填写无误，页面将跳转至“选择类型”步骤。

1.3.4　选择微信公众号类型

如果运营主体为组织，那么可以选择创建订阅公众号或服务公众号，如图 1-6 所示；如果运营主体为个人，则只能创建订阅公众号，如图 1-7 所示。

公众号只有 1 次机会可以选择成为服务号/订阅号，类型选择之后不可修改，请慎重选择。2013 年 8 月 5 日之前申请的订阅号，可以升级为服务号；在此日期之后申请的订阅号，已无法升级为服务号。

如果在这一步不确定是选择服务号还是订阅号，建议仔细阅读 1.2 节，掌握服务号和订阅号的区别，根据公众号的实际用途做出选择。

选择完公众号类型后，点击页面底部的“继续”按钮，进入注册的最后一个步骤。

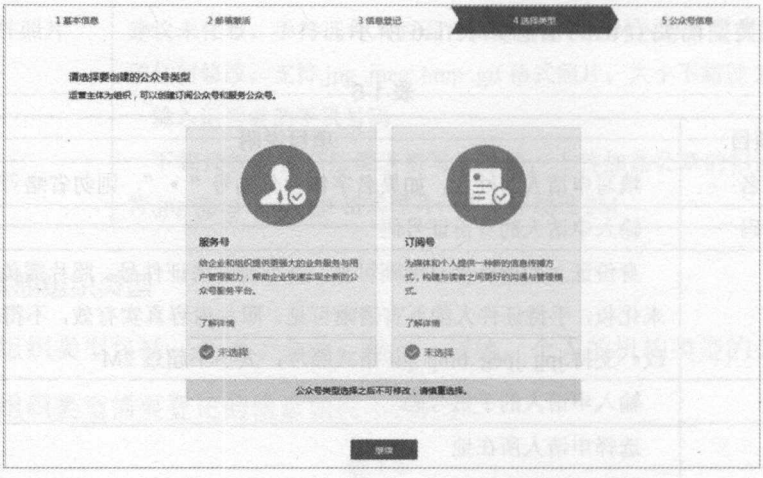


图 1-6



图 1-7

1.3.5 填写微信公众号信息

在公众号信息填写页面，需要填写公众号名称、功能介绍等信息，如图 1-8 所示。

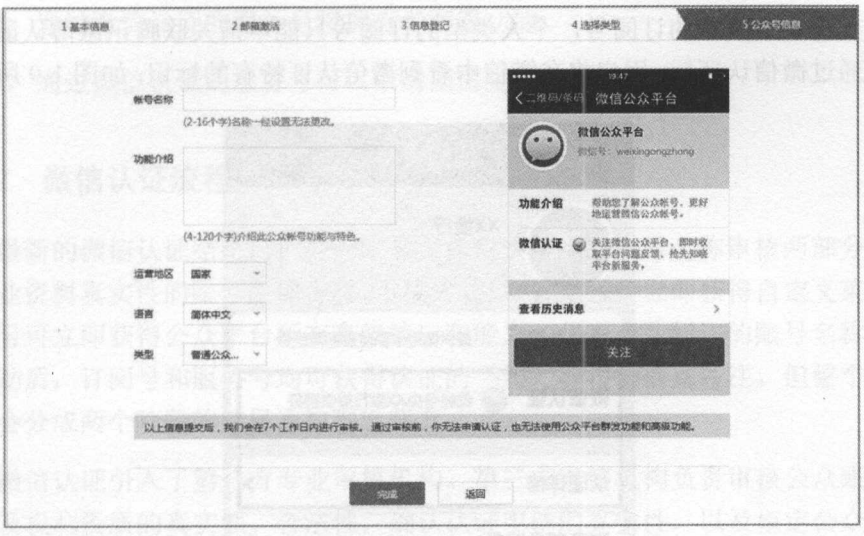


图 1-8

需要注意的是，账号名称一经设置将无法更改，而功能介绍一个月只能修改一次，这两项最终会显示在公众号的详细资料中。

公众号的账号名称是允许重复的，所以不存在抢注问题。对一些可能造成侵权的关键词，微信团队进行了保护，如“微信”、“移动”等。如果账号名称包含这类关键词，那么提交时会提示“你注册的公众号名称存在侵权风险，请先完成微博验证”，这时需要按提示进行腾讯微博验证，或避开有问题的关键词重新填写账号名称。目前微信已不支持新浪微博的验证。

在填写完公众号信息并提交后，就进入了账号审核阶段，微信团队会在 7 个工作日内完成账号审核工作。如果注册时填写的信息真实、正确，那么一般下一个工作日账号就可通过审核，正式开通。在账号通过审核前，无法使用公众平台的群发功能和高级功能，也无法申请认证。

1.4　微信认证

微信团队为了进一步规范平台运营并增强公众账号公信力，确保公众账号信息的真实性、安全性，推出了微信认证服务。微信认证过程将对公众账号身份的真实性进行验证。服务号及非个人类型的订阅号可以申请微信认证。目前微信认

证不支持个人类型的订阅号，个人类型的订阅号只能申请关联腾讯微博认证。公众号通过微信认证后，用户将在微信中看到微信认证特有的标识，如图 1-9 所示。



图 1-9

1.4.1 微信认证优势

公众号通过微信认证后会有以下优势：

◎ 通过微信认证的公众号可信度更高，更容易获取用户的信任。

◎ 通过微信的“添加朋友”搜索微信公众号时，可以按账号名称搜索到已认证的公众号，未认证的公众号只能根据微信号搜索。

◎ 通过微信的“查找公众号”搜索微信公众号时，搜索结果会优先显示已认证的公众号。

◎ 通过微信认证的订阅号或关联腾讯微博认证的订阅号都将自动开通“自定义菜单”功能。

◎ 通过微信认证的服务号将自动开通“高级接口”中的所有功能。

◎ 通过微信认证的服务号将拥有多客服功能。

◎ 通过微信认证的服务号可以申请微信支付与微信小店。

1.4.2　微信认证流程

最新的微信认证结果已拆分为账号主体资质审核和账号名称审核两部分。代表企业资料真实性的账号主体资质，审核成功后，订阅号可立即获得自定义菜单，服务号可立即获得公众平台所有高级接口和能力；代表企业标识的账号名称，审核成功后，订阅号和服务号均可获得认证的“勾”和相关信息备注。但整个认证流程会分成两个阶段的结果通知到运营者。

微信认证引入了第三方专业审核机构。第三方审核机构负责审核公众账号的主体及权利资质的真实性、合法性，确认认证申请的真实性，以及核定公众账号名称。因此，申请微信认证需支付 300 元/次的审核服务费用，每申请一次认证服务需要支付一次审核服务费。无论认证成功或失败，都需要支付审核服务费。政府及部分其他组织类型的公众账号，免收审核服务费用。

只要在规定的时间内，认证申请真实并符合相关认证规范要求，且企业/机构主体和资质材料合法、有效、完整、准确，申请人获得企业/机构的真实授权，就能通过账号资质审核。

只要在规定的时间内，认证申请的账号名称符合腾讯的命名规则，符合合法合规性的标准，就能通过账号名称审核。

账号资质审核成功后，高级功能接口权限、多客服接口等高级权限将会被保留一年。认证成功后，用户的账号名称、认证标识及认证信息将会被保留一年。因此，我们最晚应该在账号资质审核成功后一年内完成年审认证，年审认证需要另行支付审核服务费。如未通过年审认证，已开通的高级功能接口将可能会被断开，认证账号名称可能会被更改，认证标识及认证信息将会被取消。

在登录微信公众平台后，可通过单击“设置→微信认证”右侧的“开通”按钮进行微信认证申请，如图 1-10 所示。

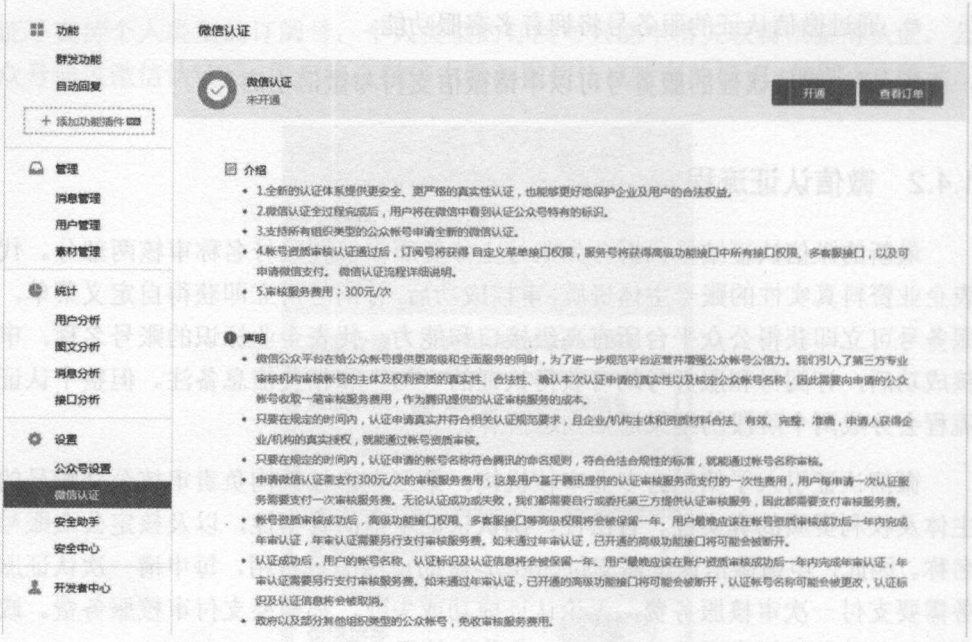


图 1-10

申请微信认证的具体步骤如下：

1. 同意协议

签署《微信公众平台认证服务协议》，勾选“同意”复选框，单击“下一步”按钮。

2. 填写资料

首先根据公众号实际运营主体选择认证主体类型，然后填写需要提交的资质材料。

认证主体类型包括企业、网店商家、事业单位媒体、其他媒体、政府（组织机构代码证上机构类型为机关法人）—认证免费、事业单位（组织机构代码证上机构类型为事业法人）、其他组织（基金会、国外政府机构驻华代表处）—认证免费、社会团体（组织机构代码证上机构类型为社团法人）、民办非企业（组织机构代码证上机构类型为民办非企业）、其他组织等。

3. 确认名称

这里需要根据页面中的说明，选择符合公众号实际情况的命名方式，部分命名方式需要按页面提示提交相关资料。如果公众号认证名称不符合微信认证的要求，那么将不能通过账号名称审核，需要修改公众号认证名称后重新提交，重新审核。微信团队对公众号认证名称的要求及命名方式如图 1-11 所示。

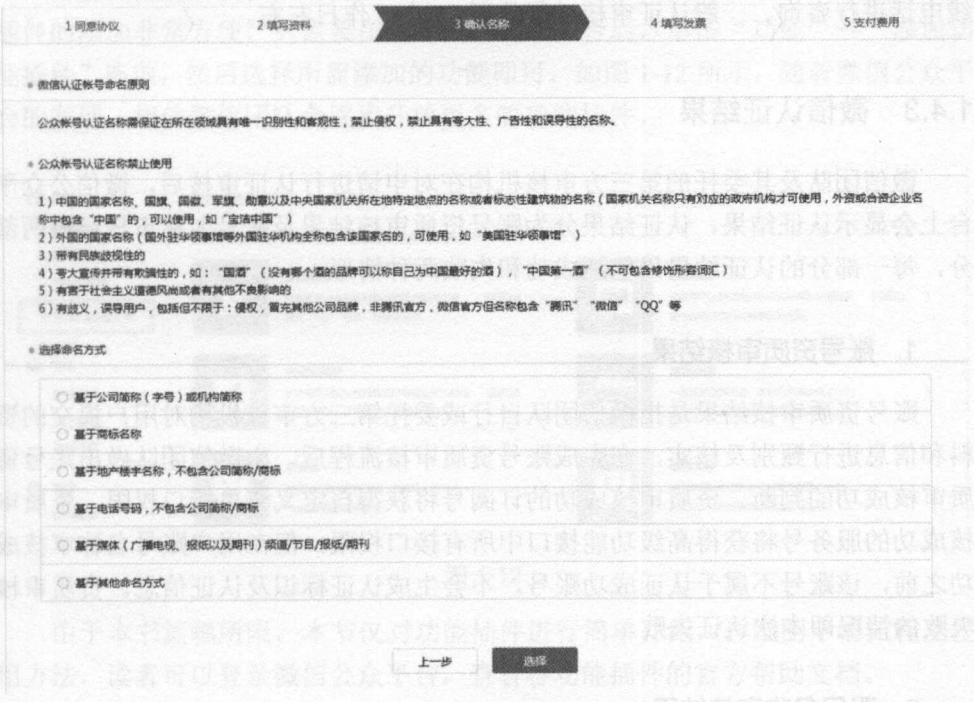


图 1-11

4. 填写发票

这里可选择开具普通发票（定额发票）、增值税专用发票。其中增值税专用发票还需提交《税务登记证》、《银行开户证明》给腾讯客服，核对资质后才能开具。开具发票需要填写发票抬头和收件地址，发票抬头为填写认证信息的企业全称。如果不需要发票，则可直接进入下一步。

5. 支付费用

最后一步是支付认证审核的费用，目前支付方式只支持微信支付。

支付完成后，即进入认证审核阶段。页面上会公布第三方审核公司的热线电话，在审核过程中该公司将有可能与公众号运营者联系沟通，核实信息或要求修改重填认证资料。如果在审核过程中遇到问题，那么我们也可以拨打该公司的热线电话进行咨询。一般认证审核过程需要 15 个工作日左右。

1.4.3 微信认证结果

微信团队及其委托的第三方审核机构在对申请进行认证审核后，微信公众平台上会显示认证结果，认证结果分为账号资质审核结果及账号名称审核结果两部分，每一部分的认证结果只包括成功和失败两种情形。

1. 账号资质审核结果

账号资质审核结果是指微信团队自行或委托第三方审核机构对用户提交的资料和信息进行甄别及核实，在完成账号资质审核流程后，由微信团队做出账号资质审核成功的判断。资质审核成功的订阅号将获得自定义菜单接口权限，资质审核成功的服务号将获得高级功能接口中所有接口权限，但在用户账号名称审核成功之前，该账号不属于认证成功账号，不会生成认证标识及认证信息。资质审核失败的情况即本次认证失败。

2. 账号名称审核结果

账号名称审核结果是指微信团队自行或委托第三方审核机构对用户提交的资料进行甄别及核实，在完成账号名称审核流程后，由微信团队做出账号名称审核成功的判断，名称审核成功的账号会生成认证标识及认证信息，名称审核失败的，将不会有认证标识及认证信息，但是，如果账号资质审核已成功的，那么就不会影响已开通的高级接口的使用了。账号名称审核成功的公众账号，会在账号详细页面展示认证标识。

1.5　功能插件简介

对于通过微信认证的服务号，除了群发消息、自动回复、自定义菜单等基础功能外，还可以通过“添加功能插件”的方式，使用微信团队开发的各种高级功能，以丰富公众号的能力和体验。

目前微信团队提供的功能插件有多客服、微信支付及微信小店等三种。功能插件的添加非常方便，只需要在登录微信公众平台后，单击“功能”→“添加功能插件”选项，然后选择所需添加的功能即可，如图 1-12 所示。随着微信公众平台的发展，相信微信团队会逐步开放更多的功能插件。

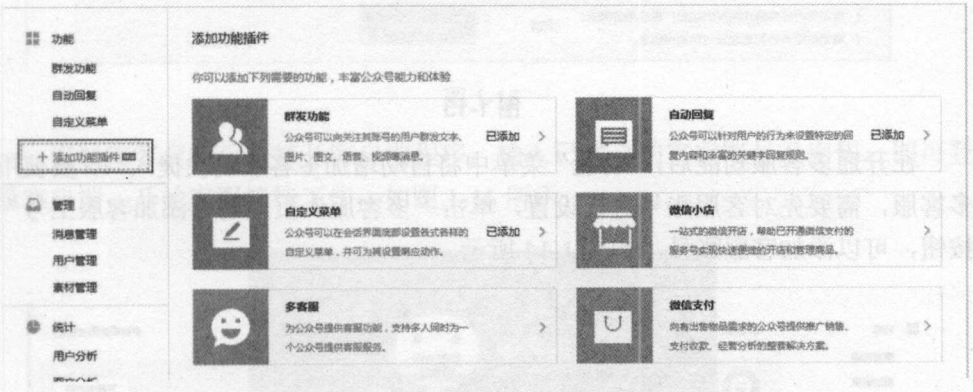


图 1-12

由于本书篇幅所限，本节仅对功能插件进行简单介绍，详细的申请流程及使用方法，读者可以登录微信公众平台，查看各功能插件的官方帮助文档。

1.5.1　多客服

多客服是微信团队在 2014 年 5 月 9 日新增加的功能插件。多客服可以为公众号提供客服功能，支持多人同时为一个公众号提供客服。

开通多客服功能，只需在“添加功能插件”页面中单击“多客服”图标，然后在“多客服功能详情”页面中单击右侧的“开通”按钮即可，如图1-13 所示。

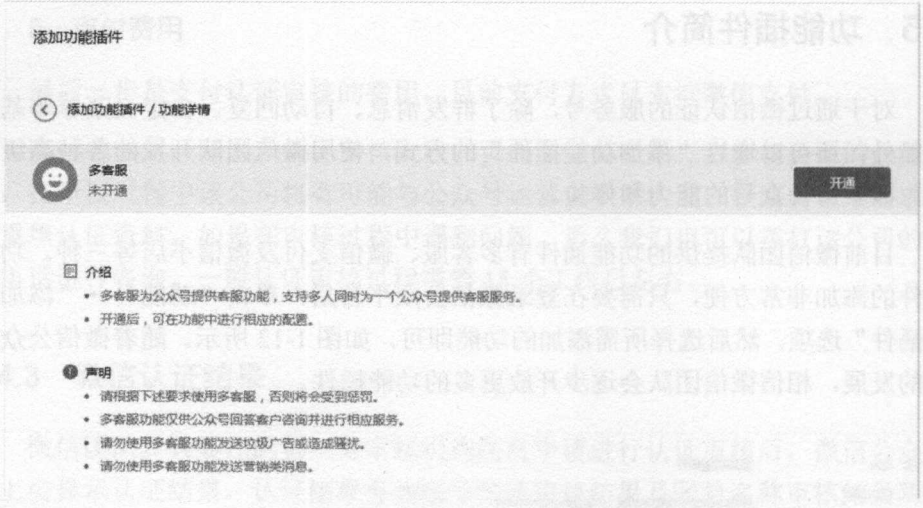


图 1-13

在开通多客服功能后，“功能”菜单中将自动增加多客服的快捷入口。要使用多客服，需要先对客服账号进行设置，单击“多客服”页面的“添加客服工号”按钮，可以添加客服账号，如图 1-14 所示。

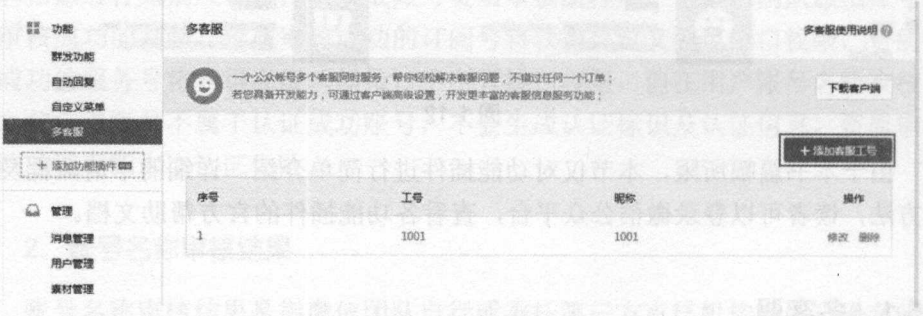


图 1-14

在填写完客服账号的工号、昵称、密码后，单击“确认添加”按钮，即可建立一个新的客服账号，如图 1-15 所示。

在建立完客服账号后，还需要下载客服使用的多客服客户端，单击“多客服”页面中的“下载客户端”按钮，即可下载，如图 1-14 所示。

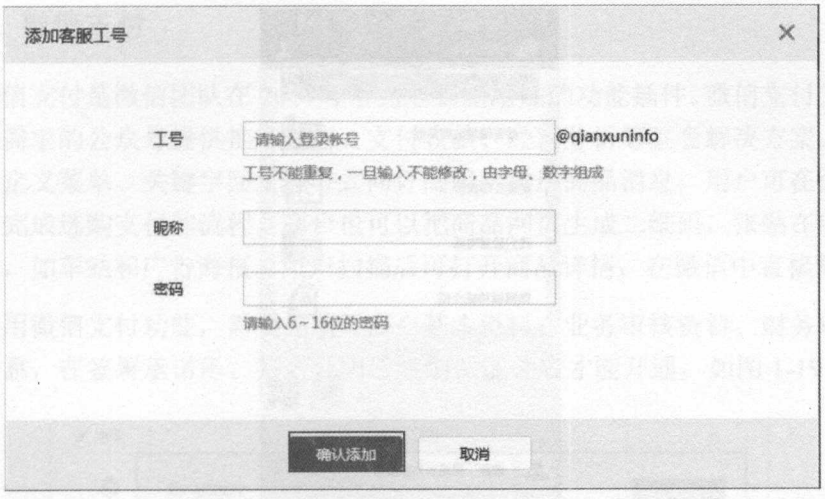


图 1-15

安装完客户端后，打开客户端程序，输入已建立的客服账号及密码，即可登录客户端，开始客服接待工作，如图 1-16 所示。



图 1-16

当微信公众号的粉丝向公众号发送消息时，如图 11-17 所示，多客服客户端将收到该消息，客服人员即可接待及回复该粉丝，如图 11-18 所示。

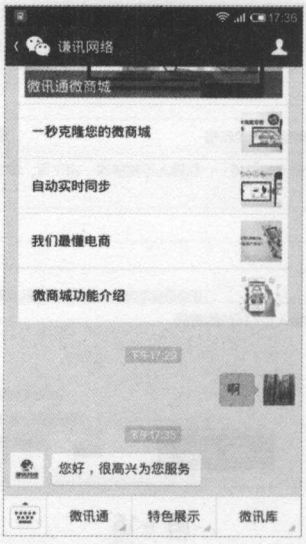


图 1-17

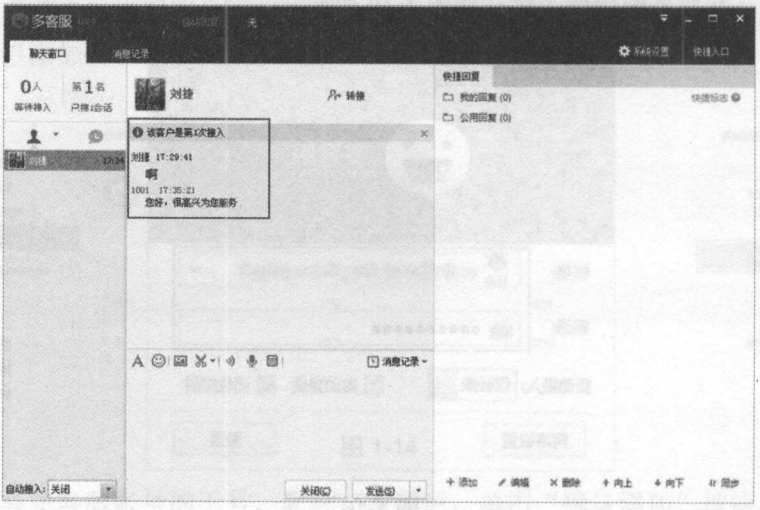


图 1-18

除了使用多客服客户端进行客服接待，多客服功能还支持在微信上进行客服接待。使用微信关注公众号“多客服助手”（微信号：duokefu），然后绑定客服账号即可。

1.5.2　微信支付

微信支付是微信团队在 2014 年 3 月 5 日新增加的功能插件。微信支付为有出售物品需求的公众号提供推广销售、支付收款、经营分析等整套解决方案。商户通过自定义菜单、关键字回复等方式向订阅用户推送商品消息，用户可在微信公众号中完成选购支付的流程。商户也可以把商品网页生成二维码，张贴在线下的场景中，如车站和广告海报。用户扫描后可打开商品详情，在微信中直接购买。

使用微信支付功能，需要先填写商户基本资料、业务审核资料、财务审核资料等信息，在签署承诺函、签署合同、缴纳保证金后才能开通，如图 1-19 所示。

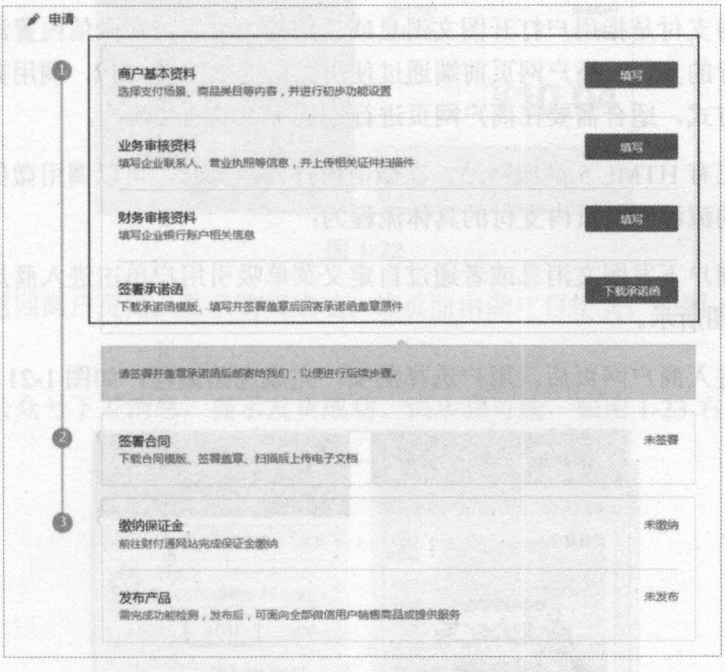


图 1-19

在开通微信支付功能后，“功能”菜单中将自动增加微信支付的快捷入口。单击“微信支付”选项，可进入微信支付管理后台，进行微信支付的设置及管理，如图 1-20 所示。

微信支付有 2 种方式：

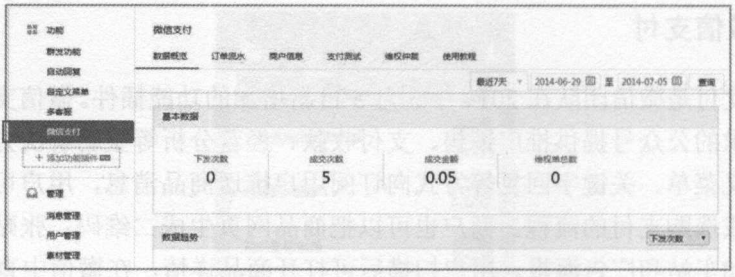


图 1-20

1. 网页内支付 JS API

网页内支付是指用户打开图文消息或者扫描二维码，在微信内置浏览器中打开网页进行的支付。商户网页前端通过使用微信提供的 JS API，调用微信支付模块。这种方式，适合需要在商户网页进行选购下单的购买流程。

商户已有 HTML 5 商城网站，在微信内打开网页时，可以调用微信支付完成下单购买的流程。网页内支付的具体流程为：

（1）商户下发图文消息或者通过自定义菜单吸引用户单击进入商户网页，如图 1-21 左图所示。

（2）进入商户网页后，用户选择购买，完成选购流程，如图 1-21 右图所示。



图 1-21

（3）调起微信支付控件，用户开始输入支付密码，如图 1-22 左图所示。

（4）密码验证通过，支付成功。商户后台得到支付成功的通知，如图 1-22 右图所示。



图 1-22

（5）返回商户页面，显示购买成功。该页面由商户自定义，如图 1-23 左图所示。

（6）公众号下发消息，提示发货成功。该步骤可选，如图 1-23 右图所示。

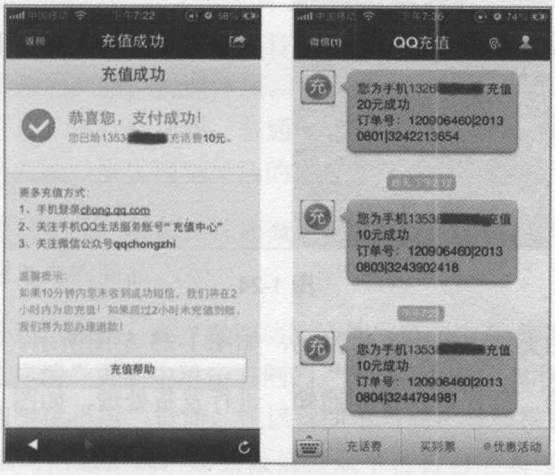


图 1-23

商户也可以把商品网页的链接生成二维码，用户扫一扫打开后即可完成购买支付。

2. 原生支付

原生支付是指商户组成符合原生支付规则的 URL 链接，用户可通过在会话中单击链接或者扫描对应的二维码直接进入微信支付模块（客户端界面），即可进行支付。这种方式，适合无需选购直接支付的购买流程。

与网页内支付场景不同，原生支付不需要经过网页选购，可以直接下单购买。原生支付的具体流程为：

（1）商户根据微信支付的规则，为不同商品生成不同的二维码，张贴在各种场景中，便于用户扫描购买，如图 1-24 左图所示。

（2）用户使用微信扫描二维码后，获取商品信息，同时到商户后台下单，如图 1-24 右图所示。



图 1-24

（3）用户开始支付，输入支付密码，如图 1-25 左图所示。

（4）支付成功，商户后台得到通知，进行发货处理，如图 1-25 右图所示。

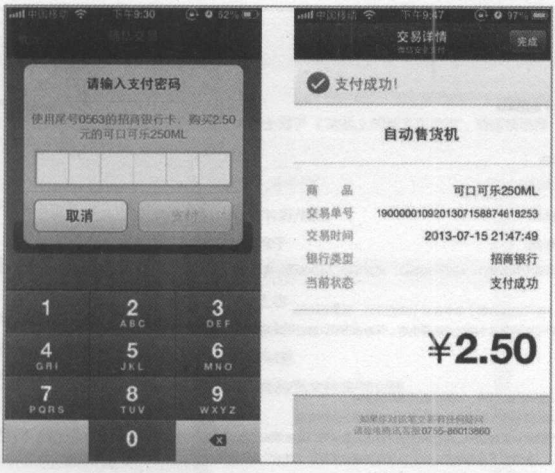


图 1-25

两种支付方式，最大的差别在于是否需要经过网页调起支付。

微信支付的申请流程及开发文档，在“微信商户服务中心”中有详细介绍。“微信商户服务中心”的网址为：https://mp.weixin.qq.com/paymch/readtemplate?t=mp/business/faq_tmpl。

1.5.3　微信小店

微信小店是微信团队在 2014 年 5 月 29 日新增加的功能插件。微信小店基于微信支付，包括添加商品、商品管理、订单管理、货架管理、维权等功能，开发者可使用接口批量添加商品，快速开店。开通微信小店后，公众号可以在微信小店中进行小店的开启、运营和使用。普通用户可直接通过小店功能管理小店，开发者则可以通过开发接口来实现更灵活的小店运营。

要申请开通微信小店，必须先开通微信支付。微信小店只可用于售卖所选的微信支付经营范围之内的商品。

微信小店的开通流程为：在“添加功能插件”页面中单击“微信小店”，然后在“微信小店功能详情”页面中单击右侧的“申请”按钮，如图 1-26 所示。页面自动跳转到“开通微信小店”页面，在该页面填写商户号和密钥，单击“提交审核”按钮，如果填写信息正确，那么微信公众平台将自动通过申请，如图 1-27 所示。

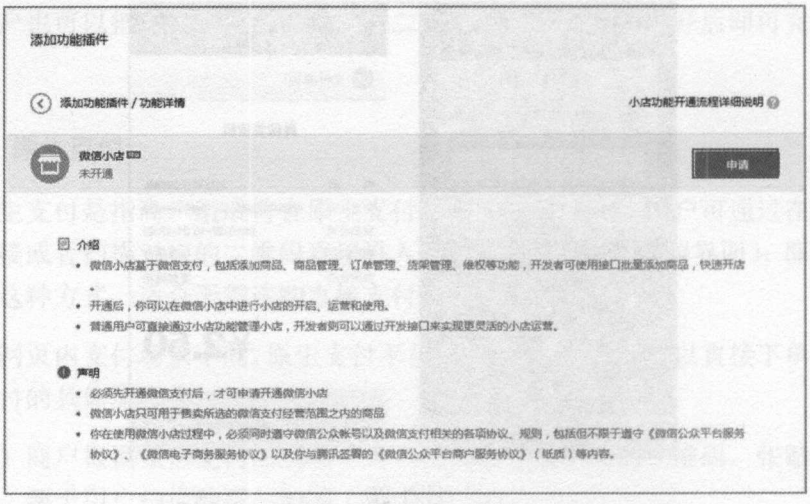


图 1-26

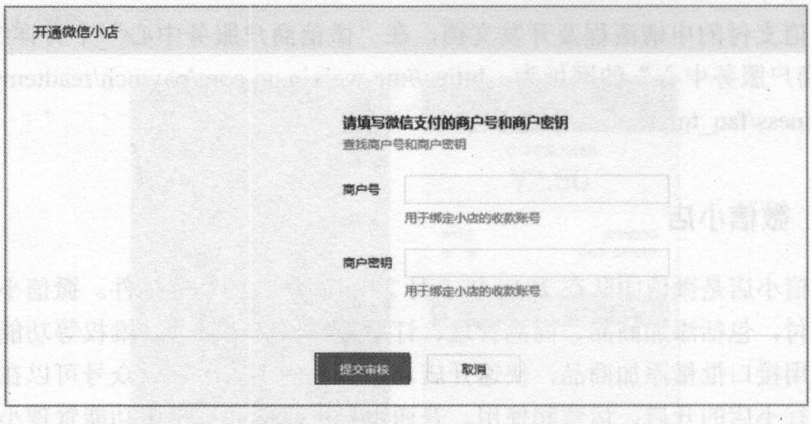


图 1-27

在开通微信小店功能后，“功能”菜单中将自动增加微信小店的快捷入口。微信小店的基本使用方法，可分为以下五个步骤：

1. 添加商品

（1）选择类目，如图 1-28 所示。

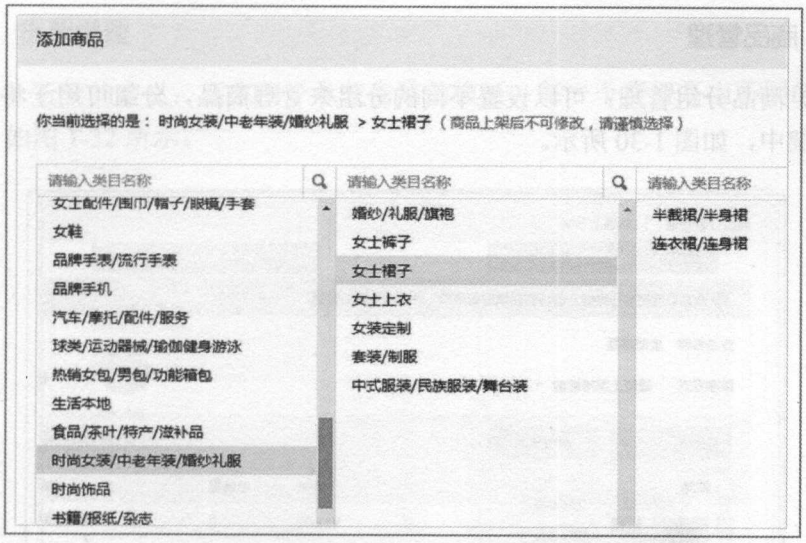


图 1-28

（2）然后再按照指引填写商品的基本信息，包括商品名称、商品图片、运费、库存、详情描述等，如图 1-29 所示。

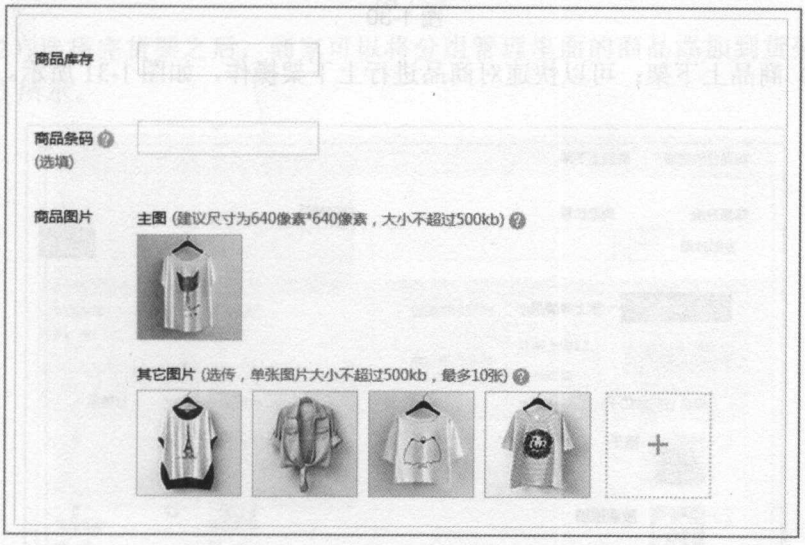


图 1-29

2. 商品管理

（1）商品分组管理：可以设置不同的分组来管理商品，分组可用于将商品填充到货架中，如图 1-30 所示。

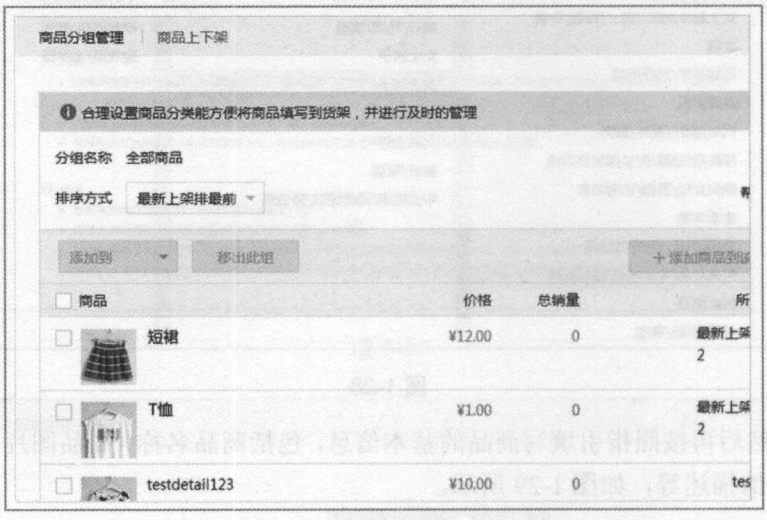


图 1-30

（2）商品上下架：可以快速对商品进行上下架操作，如图 1-31 所示。

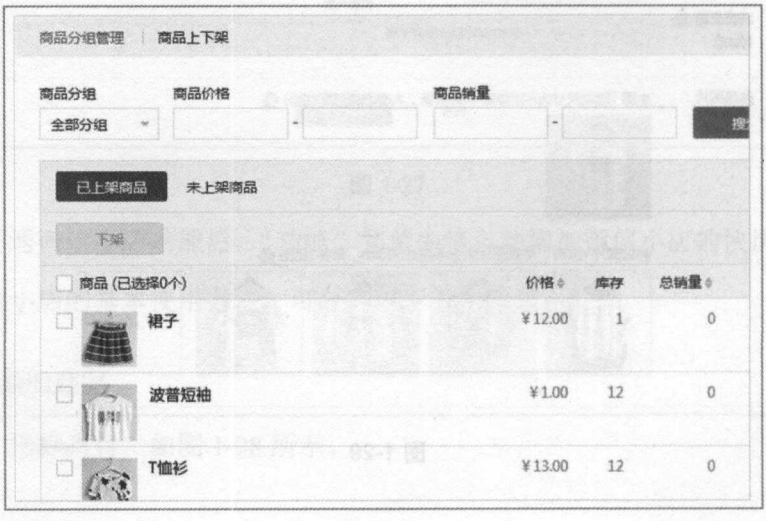


图 1-31

3. 货架管理

（1）货架的定义：商家用于承载商品的模板，每一个货架是由不同的控件组成的，如图 1-32 所示。

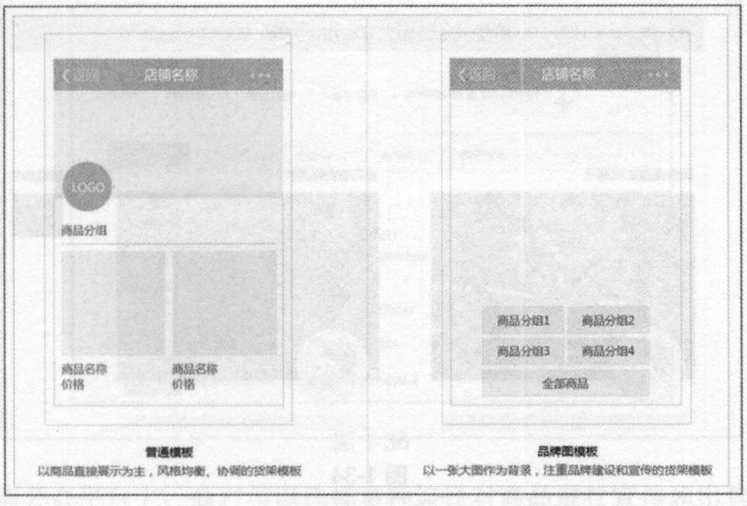


图 1-32

（2）选择完货架之后，商家可以将分组管理里面的商品添加到货架中，如图 1-33 所示。

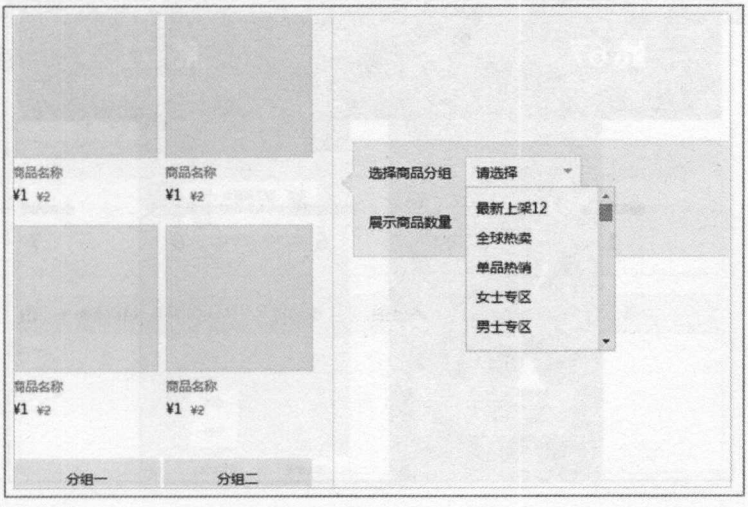


图 1-33

（3）发布货架：将编辑好的货架单击发布，然后复制链接，链接可以填入自定义菜单中，或者下发到商品消息中，如图 1-34 所示。



图 1-34

3. 小店概况

可以查看小店所有的数据信息：订单数、成交量等，如图 1-35 所示。

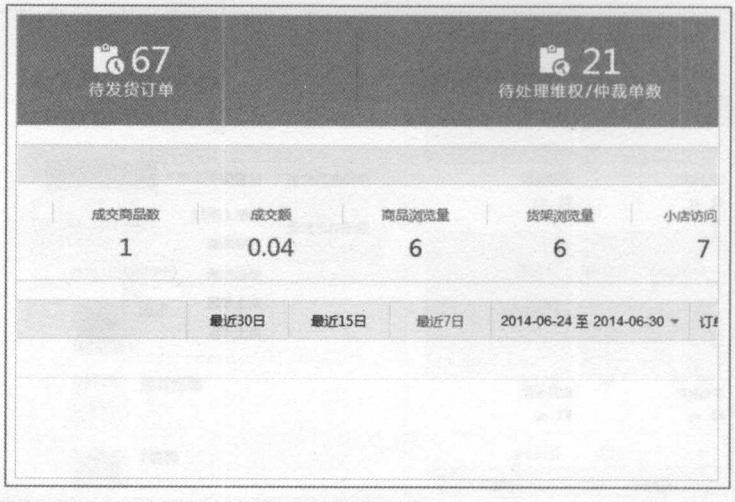


图 1-35

4. 订单管理

用户支付成功会生成一笔订单，商家可以查询订单，并进行发货等操作，如图 1-36 所示。

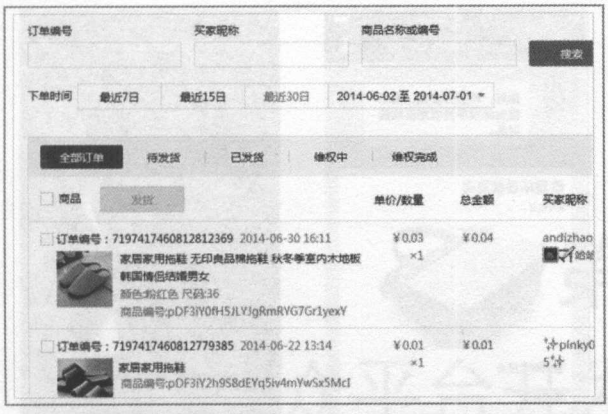


图 1-36

在微信公众平台上，商户可以选择多种途径对商品进行宣传和出售，比如通过自定义菜单、消息下发等方式实现，而用户则可以实现灵活购买。

5. 自定义菜单

商户可以在自定义菜单中分类列出需要出售的商品和购买入口，还可以按照自己的运营模式进行维护，如图 1-37 所示。

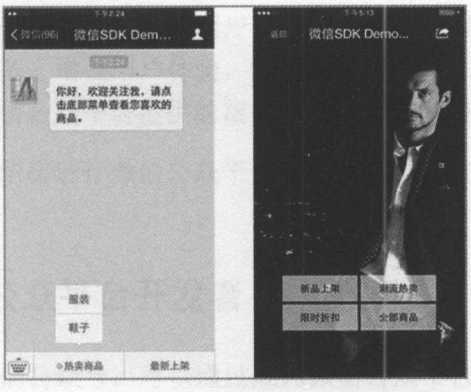


图 1-37

6. 公众号消息下发

商户可通过下发图文消息、设置关键字回复等方式让客户获取商品，如图 1-38 所示。



图 1-38

微信小店的功能介绍、使用方法及接口文档，在“微信公众平台—小店接入流程指引”中有详细介绍。具备开发能力的用户，可基于接口实现更灵活的功能。“微信公众平台—小店接入流程指引”的网址为：https://mp.weixin.qq.com/cgi-bin/readtemplate?t=shop/faq_tmpl。

第 2 章
微信公众平台开发准备

通过阅读第 1 章的内容，相信读者已经对微信公众平台比较熟悉，并且开始了公众账号的申请与认证流程。

公众账号申请成功后，用户就可以关注该微信公众号了，但此时的公众账号只能使用微信公众平台的默认功能为用户提供一些简单的服务。要想为用户提供优质的、个性化的服务，增强用户体验，提高用户忠诚度，提升微信公众号的运营效果，只有使用微信公众平台提供的接口，开发各种高级应用才能满足。

本章将为读者介绍在进行微信公众平台开发前，需要进行的一些准备工作。

2.1 成为微信公众平台开发者

对于已注册的微信公众号，可以通过成为微信公众平台的开发者，来为微信公众平台的开发做好准备。

微信公众账号注册成功后，需要先成为开发者，才能使用微信公众平台提供的接口进行开发。

首先登录微信公众平台，选择“开发者中心”，勾选“我同意《微信公众平台开发者服务协议》”复选项，单击“成为开发者”按钮，如图 2-1 所示。

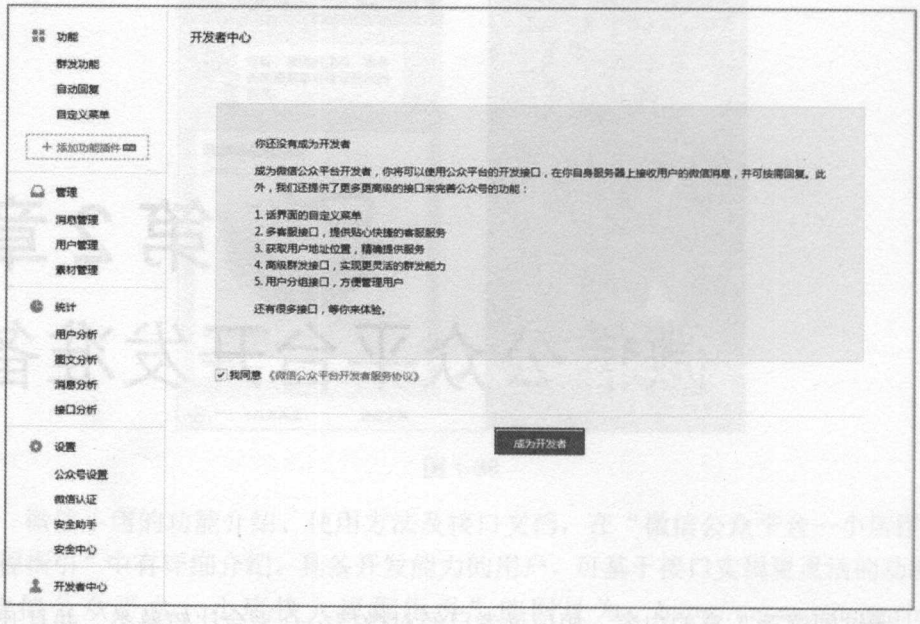


图 2-1

如果微信公众号的账号信息不完整，微信公众平台会要求先补全账号信息才能成为开发者，如图 2-2 所示。需要补全的账号信息包括公众号头像、微信号、所在地址等，在“设置”→“公众号设置”中进行填写。

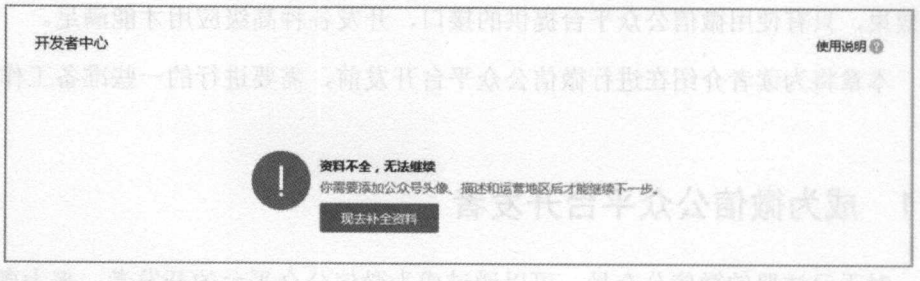


图 2-2

补全账号信息后，重新进入“开发者中心”，再次单击“成为开发者”按钮，即可成为微信公众平台的开发者，并进入“填写服务器配置”页面，如图 2-3 所示。

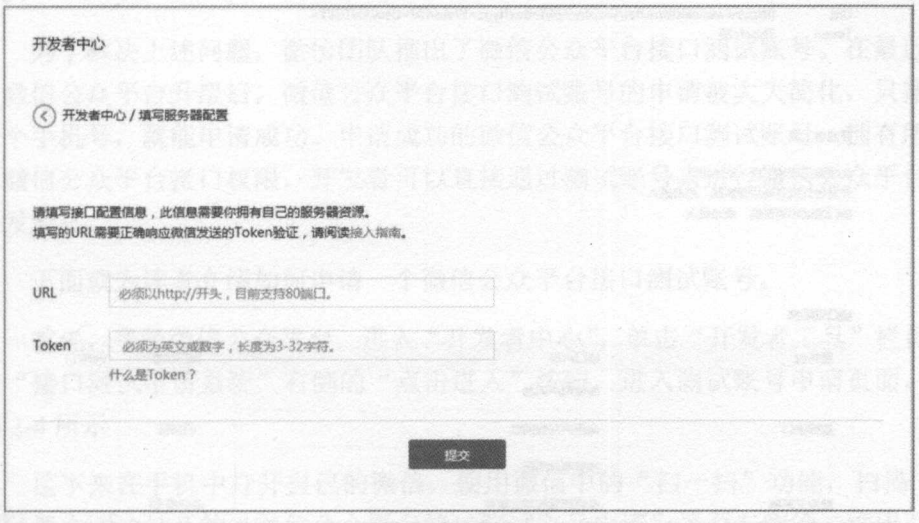


图 2-3

在“填写服务器配置”页面需要填写 URL 和 Token 两个值。URL 指的是能够接收处理微信服务器发送的 GET/POST 请求的网址，该网址应该已部署到开发者的 Web 服务器中，能被公网访问到。Token 是用作微信服务器与开发者 Web 服务器之间进行通信的安全签名，每次微信服务器和开发者 Web 服务器的通信都会含有该 Token 值经过加密后的信息，通过对比 Token 值，验证通信的安全性。Token 用于防止未授权的第三方伪造身份、窃取信息，请勿泄露 Token。

填写完 URL 和 Token，单击“提交”按钮，微信服务器会向填写的 URL 发起 GET 请求进行校验。如果校验成功，那么页面顶部会弹出“提交成功”的提示信息，然后自动跳转到“开发者中心”，在页面中显示前一步配置的 URL 和 Token，以及开发者工具和获取的接口权限列表，如图 2-4 所示。如果校验失败，那么请检查 URL 和 Token 填写是否有误，URL 是否能正确地被公网访问到。

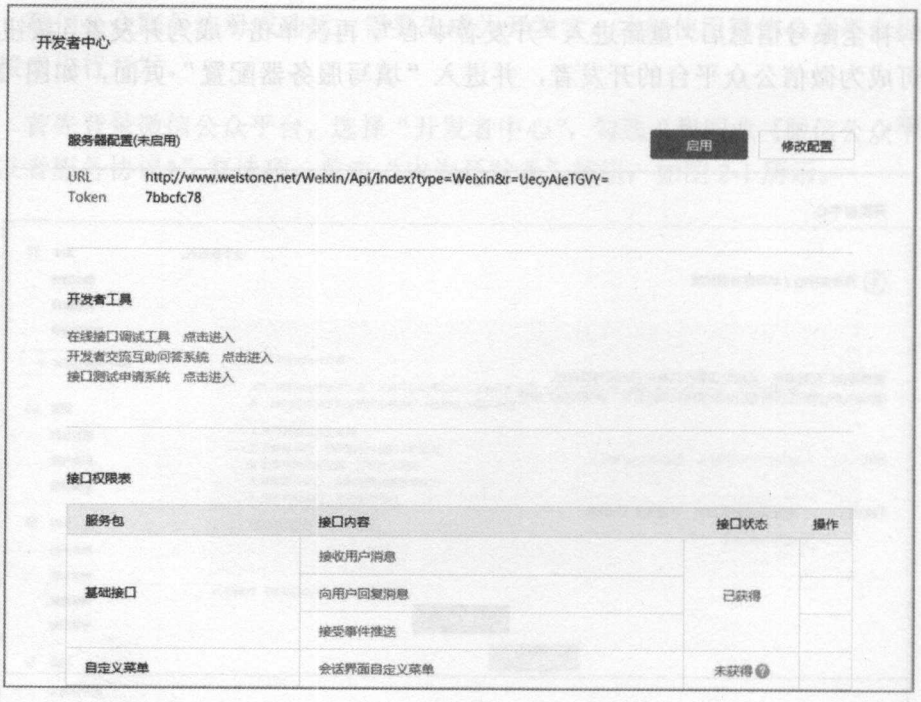


图 2-4

最后，单击“启用”按钮，开启服务器配置。开启后，页面将显示“服务器配置（已启用）”字样，如图 2-5 所示。

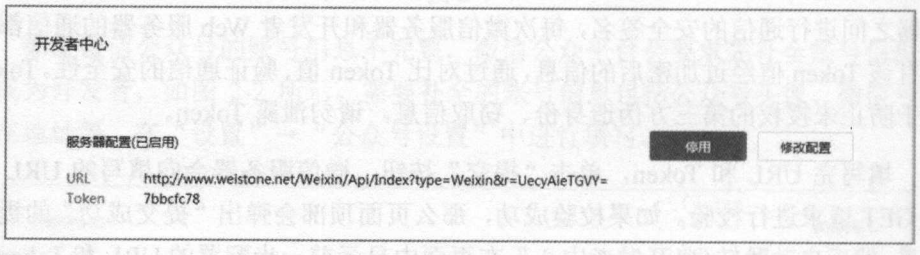


图 2-5

2.2 微信公众平台接口测试账号申请

由于除了认证服务号以外的其他类型公众账号，并没有 10 大高级接口的权限。

对于刚注册的微信公众号，申请微信认证需要较长的时间，在通过微信认证之前将无法开发需要调用高级接口的功能。同时，对于认证服务号来说，如果直接在运营中的微信公众号上进行开发测试，势必造成对关注者的打扰和用户体验的下降。

为了解决上述问题，微信团队推出了微信公众平台接口测试账号。在最近一次微信公众平台升级后，微信公众平台接口测试账号的申请被大大简化，只需要一个手机号，就能申请成功。申请成功的微信公众平台接口测试账号，拥有所有的微信公众平台接口权限，开发者可以直接通过测试账号来进行微信公众平台的开发测试。

下面就为读者介绍如何申请一个微信公众平台接口测试账号。

首先，登录微信公众平台，进入“开发者中心”，单击“开发者工具”栏目中的“接口测试申请系统”右侧的“点击进入”按钮，进入测试账号申请页面，如图 2-4 所示。

接下来在手机中打开自己的微信，使用微信中的“扫一扫”功能，扫描 PC 浏览器中刚才打开的“微信公众平台接口测试账号申请”页面左侧的二维码，如图 2-6 所示。



图 2-6

扫描二维码成功后，手机上会出现“应用登录”页面，单击下方的“确认登录”按钮，如图 2-7 所示。

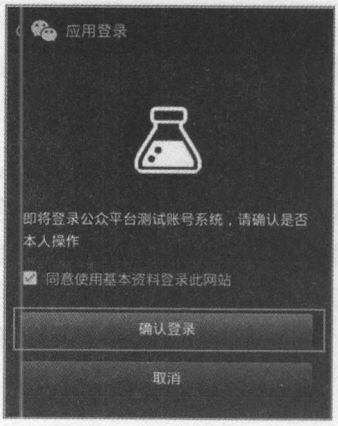


图 2-7

这时，PC 浏览器会自动使用扫描二维码的微信号登录到“管理测试号”页面，如图 2-8 所示。在“接口配置信息”栏目中要填写 URL 和 Token 两个值，输入完毕后，单击“提交”按钮。微信服务器如果校验成功，则页面顶部会弹出“提交成功”的提示信息，这表示我们的微信公众平台测试账号已注册成功。测试账号拥有的接口权限在页面底部的“体验接口权限表”栏目中有详细说明，如图 2-9 所示。



图 2-8



图 2-9

需要注意的是，一个微信个人用户只能申请一个微信公众平台接口测试账号。目前测试账号的有效期为首次申请成功后一年，过期需要重新申请。测试账号的有效期在“管理测试号”页面顶部有明确提示，如图 2-8 所示。每个测试账号只能绑定 20 个微信个人用户用于测试。测试账号绑定的微信个人用户可以在“测试号二维码”栏目中查看用户列表及进行移除操作，如图 2-10 所示。



图 2-10

每次登录测试账号管理后台，都需要使用注册测试账号时使用的微信，扫描“微信公众平台接口测试账号申请”页面左侧的二维码，然后在微信中单击“确认登录”按钮来登录。

测试账号申请成功后，在手机中用微信扫描“测试号二维码” 栏目中的二维码，即可关注测试账号，如图 2-11 所示。



图 2-11

第3章
实现URL接入

无论是微信公众平台接口测试账号还是正式的微信公众号，进行微信公众平台开发的第一步都是要实现URL接入，使“填写服务器配置”页面中填写的URL能正常通过微信服务器的校验。本章将为读者介绍如何实现URL接入。

3.1 接口校验方法

要使“填写服务器配置”页面中填写的URL能正常通过微信服务器的校验，需要按《微信公众平台开发者文档》中“接入指南”的要求，编写接口校验方法。

开发者提交信息后，微信服务器将发送GET请求到填写的URL上，GET请求携带四个参数，如表3-1所示。

表 3-1

参　数	描　述
signature	微信加密签名，signature 结合了开发者填写的 token 参数和请求中的 timestamp 参数、nonce 参数
timestamp	时间戳
nonce	随机数
echostr	随机字符串

开发者通过检验 signature 对请求进行校验（下面有校验方式）。若确认此次 GET 请求来自微信服务器，请原样返回 echostr 参数内容，则接入生效，成为开发者，否则接入失败。

加密/校验流程如下：

1. 将 token、timestamp、nonce 三个参数进行字典排序。

2. 将三个参数字符串拼接成一个字符串进行 sha1 加密。

3. 开发者获得加密后的字符串可与 signature 对比，标识该请求来源于微信。

在 C#中，可以使用 LINQ 来对参数进行字典排序，代码如下：

```
string[] arr = new[] { token, timestamp, nonce }.OrderBy(z => z).ToArray();
```

将三个参数字符串进行拼接：

```
string arrString = string.Join("", arr);
```

sha1 加密可以使用.NET Framework 提供的 System.Security.Cryptography.SHA1 类来实现：

```
System.Security.Cryptography.SHA1 sha1 =
System.Security.Cryptography.SHA1.Create();
byte[] sha1Arr = sha1.ComputeHash(Encoding.UTF8.GetBytes(arrString));
```

sha1 加密后的结果是一个 byte 数组，因此，还需要将加密结果转换为字符串：

```
StringBuilder enText = new StringBuilder();
foreach (var b in sha1Arr)
{
    enText.AppendFormat("{0:x2}", b);
}
```

3.2　实现接口校验程序

接下来以使用 Microsoft Visual Studio 2012 开发环境，构建一个 ASP.NET Web 应用程序为例，向读者详细介绍开发接口校验程序的完成步骤。

1．新建一个名为 Sample_1 的 ASP.NET 空 Web 应用程序，如图 3-1 所示。

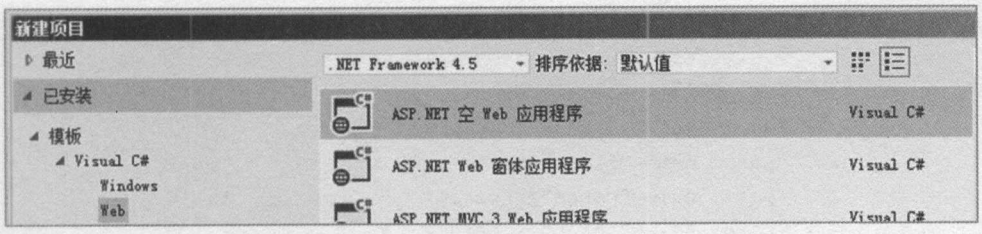


图 3-1

2．在项目下创建一个 CheckSignature 类，该类对接口校验方法进行了封装，调用名为 Check 的静态方法，即可实现接口校验。CheckSignature.cs 的完整代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Sample_1
{
    public class CheckSignature
    {
        /// <summary>
        /// 网站没有提供 Token（或传入为 null）情况下的默认 Token
        /// </summary>
        public const string Token = "weixin";

        /// <summary>
        /// 检查签名是否正确
        /// </summary>
        /// <param name="signature"></param>
        /// <param name="timestamp"></param>
```

```
        /// <param name="nonce"></param>
        /// <param name="token"></param>
        /// <returns></returns>
        public static bool Check(string signature, string timestamp, string
nonce, string token)
        {
            return signature == GetSignature(timestamp, nonce, token);
        }

        /// <summary>
        /// 返回正确的签名
        /// </summary>
        /// <param name="timestamp"></param>
        /// <param name="nonce"></param>
        /// <param name="token"></param>
        /// <returns></returns>
        public static string GetSignature(string timestamp, string nonce,
string token = Token)
        {
            token = token ?? Token;
            string[] arr = new[] { token, timestamp, nonce }.OrderBy(z =>
z).ToArray();
            string arrString = string.Join("", arr);
            System.Security.Cryptography.SHA1 sha1 = System.Security. Cryptography.
SHA1.Create();
            byte[] sha1Arr = sha1.ComputeHash(Encoding.UTF8.GetBytes(arrString));
            StringBuilder enText = new StringBuilder();
            foreach (var b in sha1Arr)
            {
                enText.AppendFormat("{0:x2}", b);
            }
            return enText.ToString();
        }
    }
}
```

3．在项目下创建一个名为 Index.aspx 的 Web 窗体，用于接收微信服务器 GET 请求传递的四个校验参数，调用 CheckSignature.Check 方法进行接口校验。如果校验通过，则原样返回 GET 参数中的 echostr。同时，为了方便判断网址是否能正确访问，加入了校验失败的提示信息。Index.aspx.cs 的完整代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Sample_1
{
    public partial class Index : System.Web.UI.Page
    {
        //与微信公众账号后台的 Token 设置保持一致，区分大小写
        private readonly string Token = "weixin";

        protected void Page_Load(object sender, EventArgs e)
        {
            Auth();
        }

        /// <summary>
        /// 处理微信服务器验证消息
        /// </summary>
        private void Auth()
        {
            string signature = Request["signature"];
            string timestamp = Request["timestamp"];
            string nonce = Request["nonce"];
            string echostr = Request["echostr"];
            if (Request.HttpMethod == "GET")
            {
                //get method - 仅在微信后台填写 URL 验证时触发
                if (CheckSignature.Check(signature, timestamp, nonce, Token))
                {
                    WriteContent(echostr); //返回随机字符串则表示验证通过
                }
                else
                {
                    WriteContent("failed:" + signature + "," + CheckSignature.
GetSignature(timestamp, nonce, Token) + "。" +"如果你在浏览器中看到这句话，说明此
地址可以被作为微信公众账号后台的 URL，请注意保持 Token 一致。");
```

```
            }
            Response.End();
        }
    }

    private void WriteContent(string str)
    {
        Response.Output.Write(str);
    }
  }
}
```

Sample_1 项目完整的结构如图 3-2 所示。

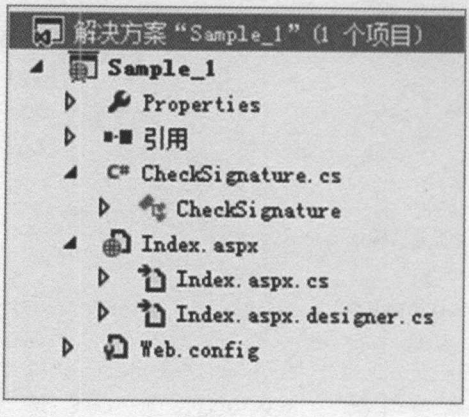


图 3-2

3.3 本地测试

使用 Microsoft Visual Studio 2012 打开 Sample_1 项目，按“F5”键启动项目调试，会自动打开浏览器。如果看到如图 3-3 所示的界面，则表明程序能被正常访问，也能正确处理 GET 请求。

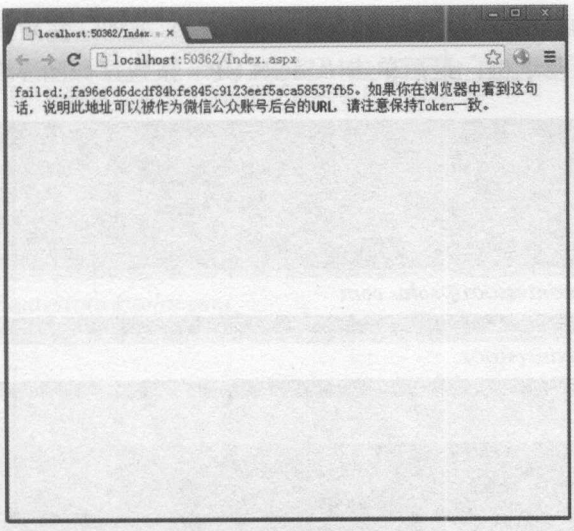


图 3-3

3.4　使用 AppHarbor 的部署接口校验程序

AppHarbor 是一个提供.NET 服务端运行环境的云平台，能帮助开发者快速部署公网应用。AppHarbor 提供了丰富、全面的.NET 云环境解决方案，想要全面了解 AppHarbor，绝不是一件容易的事。不过，如果只是将我们已经开发好的项目部署到 AppHarbor 上，那就简单多了。在 AppHarbor 上部署.NET 应用主要分为三个步骤：注册账号、创建应用，以及通过 Git 上传项目。

注：有关 AppHarbor 更详细的介绍，请访问其官方入门教程，网址为：“http://support.appharbor.com/kb/getting-started”。

1. 注册账号

首先打开 AppHarbor 官网：“https://appharbor.com/”，单击右上角的“Sign Up”按钮，打开注册账号页面，如图 3-4 所示。



图 3-4

在注册账号页面中依次填写邮箱地址、用户名、密码，然后勾选“I agree to the terms of service”复选框，最后单击“SIGN UP”按钮，如图 3-5 所示。

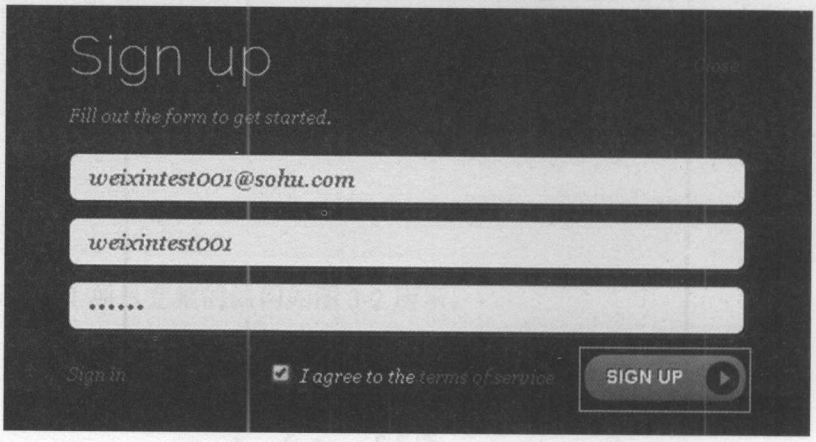


图 3-5

如果注册成功，则会在首页显示如图 3-6 所示的注册成功提示，要求我们去注册时填写的邮箱激活账号。如果注册失败，请更换邮箱和用户名重新注册。

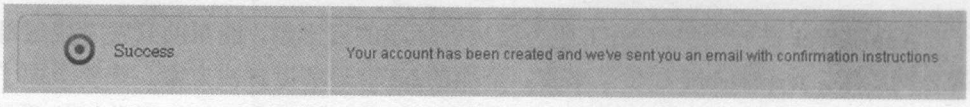


图 3-6

接下来登录注册时填写的邮箱，打开 AppHarbor 发送的激活邮件，单击“verify your email address”链接激活账号，如图 3-7 所示。

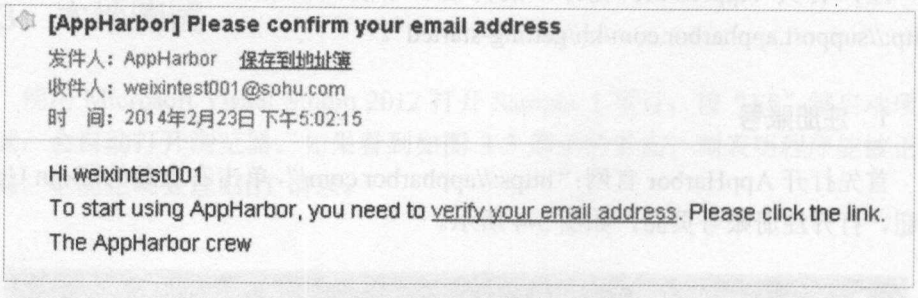


图 3-7

如果账号激活成功，则页面会跳转到登录页面。此时，填入注册时填写的邮箱地址或用户名及密码，就能登录到 AppHarbor 云平台了，如图 3-8 所示。

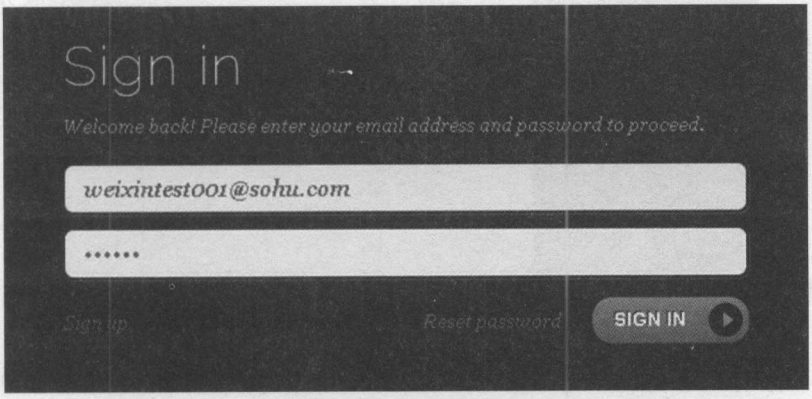


图 3-8

2. 创建应用

在成功注册账号和登录 AppHarbor 后，需要先创建一个应用才能将我们的项目部署到 AppHarbor 上。AppHarbor 创建的第一个应用是免费使用的，如果需要创建多个应用，则需要支付相应的费用。

在登录成功后，会进入如图 3-9 所示的界面，在此界面中输入要创建的应用名称，这里我们将应用取名为“weixintest”，然后单击“CREATE NEW”按钮创建一个新的应用。

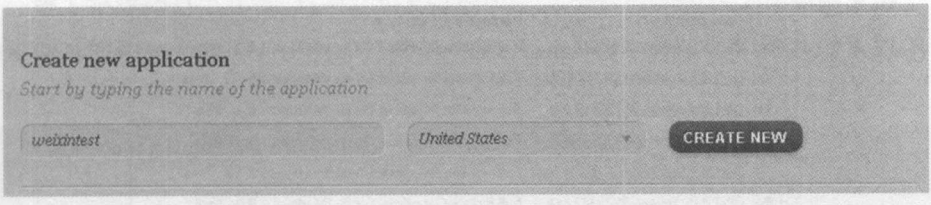


图 3-9

如果应用创建成功，则会进入如图 3-10 所示的界面；如果创建失败，那么请修改应用名称，重新创建应用。

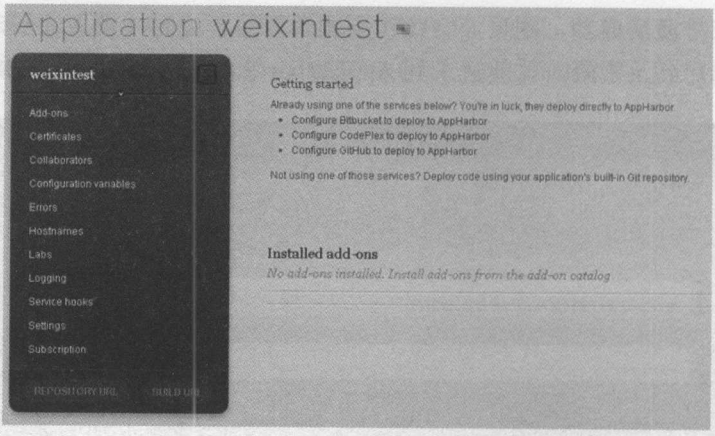


图 3-10

3. 通过 Git 上传项目

AppHarbor 使用分布式版本控制工具 Git 来上传项目。Git 是目前世界上使用最广泛的跨平台版本控制系统，由于篇幅所限，在此仅介绍如何使用 Git 上传我们的项目到 AppHarbor。有关 Git 的详细介绍，可以在百度百科中搜索 Git 进行查阅，网址为：http://baike.baidu.com/subview/1531489/12032478.htm?fr=aladdin。

首先，我们要下载 Git 的 Windows 版工具 msysgit，下载地址为："https://code.google.com/p/msysgit/downloads/list?q=full+installer+official+git"。我们选择 msysgit 的最新版本"Git-1.9.0-preview20140217.exe"进行下载，如图 3-11 所示。

Filename ▼	Summary + Labels ▼
Git-1.9.0-preview20140217.exe	Full installer for official Git for Windows 1.9.0 Beta Featured
Git-1.8.5.2-preview20131230.exe	Full installer for official Git for Windows 1.8.5.2 Beta
Git-1.8.4-preview20130916.exe	Full installer for official Git for Windows 1.8.4 Beta
Git-1.8.3-preview20130601.exe	Full installer for official Git for Windows 1.8.3 Beta
Git-1.8.1.2-preview20130201.exe	Full installer for official Git for Windows 1.8.1.2 Beta
Git-1.8.0-preview20121022.exe	Full installer for official Git for Windows 1.8.0 Beta
Git-1.7.11-preview20120710.exe	Full installer for official Git for Windows 1.7.11 Beta
Git-1.7.11-preview20120704.exe	Full installer for official Git for Windows 1.7.11 Beta
Git-1.7.10-preview20120409.exe	Full installer for official Git for Windows 1.7.10 Beta
Git-1.7.9-preview20120201.exe	Full installer for official Git for Windows 1.7.9 Beta
Git-1.7.8-preview20111206.exe	Full installer for official Git for Windows 1.7.8 Beta

图 3-11

在下载完成后，打开“Git-1.9.0-preview20140217.exe”安装程序，按照默认选项进行安装。安装完毕后，单击“开始”→“所有程序”→“Git”→“Git Bash”，启动 Git 命令行工具，如图 3-12 所示。

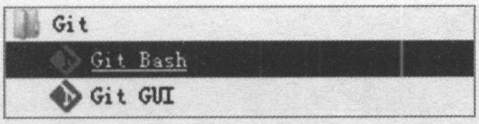


图 3-12

接下来，我们需要先创建一个文件夹用于存放要发布到 AppHarbor 上的项目文件。这里选择“D:/weixintest”文件夹。然后将需要上传的项目文件复制到此文件夹中。Sample_1 项目需要上传的项目文件有：bin 文件夹、Index.aspx 文件、Web.config 文件，如图 3-13 所示。

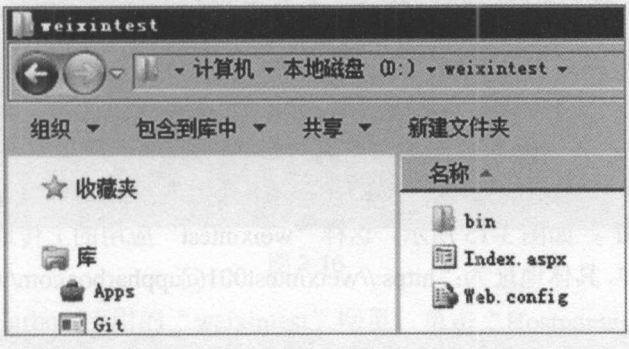


图 3-13

接下来将项目上传到 AppHarbor 应用中。首先需要初始化 Git 的版本库，并创建项目的第一个版本，在刚才打开的“Git Bash”命令行工具中依次输入以下命令：

```
cd d:/weixintest/
git init
git add .
git commit -m "Initial commit"
```

执行结果如图 3-14 所示。

如果在输入命令 git commit -m "Initial commit"后出现“*** Please tell me who you are.”的提示，则需要先设置自己的 Git 账号信息才能继续，请依次输入以下

命令后再重新执行 git commit -m "Initial commit"：

```
git config --global user.email "weixintest001@sohu.com"
git config --global user.name "weixintest001"
```

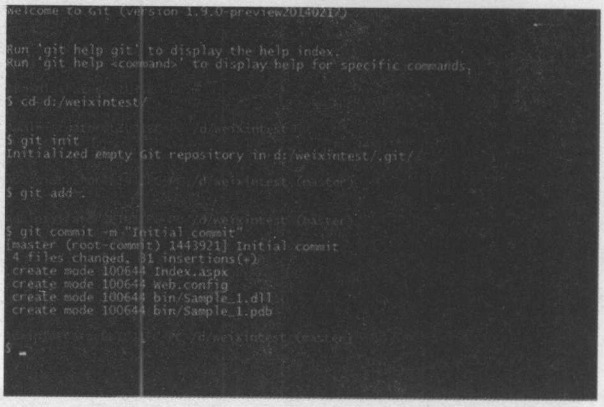


图 3-14

下一步，我们需要将刚创建的 Git 版本上传到前面创建的 AppHarbor 应用“weixintest”中。首先回到我们刚创建成功的 AppHarbor 应用“weixintest”的网页，单击左下角的“REPOSITORY URL”按钮，会出现页面提示“Copied repository URL to clipboard”，如图 3-15 所示，这样“weixintest”应用的上传地址就已经被保存到了剪切板中，具体地址为：“https://weixintest001@appharbor.com/weixintest-1.git”。

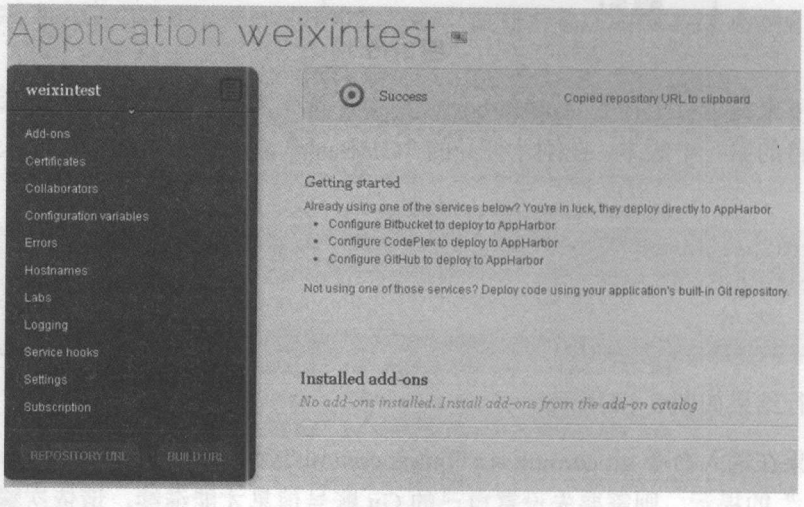


图 3-15

然后，在“Git Bash”命令行工具中输入以下命令：

```
    git remote add appharbor https://weixintest001@appharbor.com/weixintest-1.
git
    git push appharbor master
```

此时会有提示要求输入密码“Password for 'https://weixintest001@appharbor.com':”，输入 AppHarbor 账号的登录密码并按回车键，稍等几秒钟，上传成功的提示信息会显示出来。这样我们就成功地将项目上传到了 AppHarbor 的应用中，如图 3-16 所示。

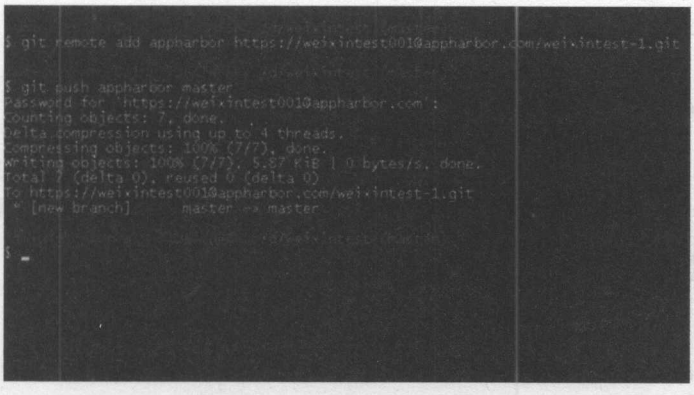


图 3-16

回到 AppHarbor 应用的“weixintest”网页，单击“Hostnames”链接，会显示出我们应用的域名 http://weixintest-1.apphb.com/，如图 3-17 所示。

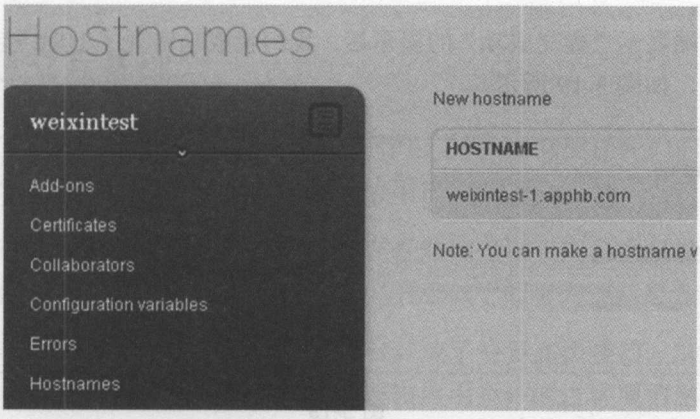


图 3-17

在浏览器中打开网址“http://weixintest-1.apphb.com/index.aspx”，我们将看到图3-3中的提示信息，这表明我们的接口校验程序已能被微信服务器正常访问到，也能正确处理 GET 请求。

3.5 接入微信公众平台

在微信公众平台“开发者中心”的“服务器配置”中，填入我们的接口校验程序 URL：“http://weixintest-1.apphb.com/index.aspx”，以及我们在接口校验程序中设置的 Token：“weixin”，并提交，如图 3-18 所示。

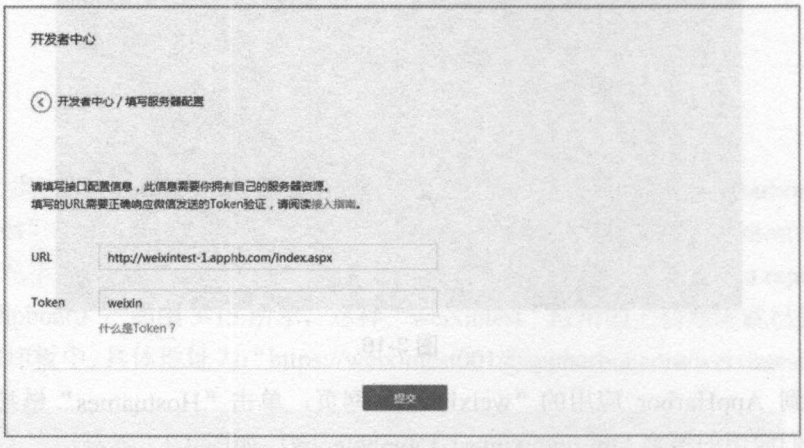


图 3-18

页面顶端显示“提交成功”的提示后，我们就成功实现了 URL 接入微信公众平台的工作，如图 3-19 所示。

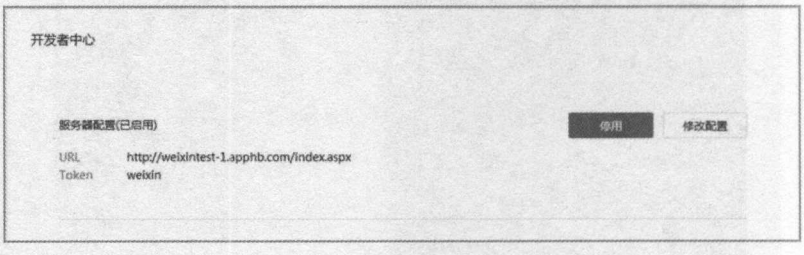


图 3-19

第 4 章
微信公众平台消息处理框架

本章主要讲解如何在 Microsoft Visual Studio 2012 开发环境中，使用 C#开发语言开发一套与微信公众平台消息接口实现对接的“微信公众平台消息处理框架”。

4.1 消息交互基础

在实现了 URL 接入微信公众平台的工作后，下一步需要让我们的程序能正确接收用户发送的微信消息，以及发送消息给用户。

传统的微信公众平台消息交互，只有在普通微信用户向公众号发送消息后，公众号才能回复消息给普通微信用户，这类消息以下简称为“消息”。

在微信 5.0 版本推出后，微信公众平台开放了许多高级接口。其中的客服消息接口，允许微信公众号主动发消息给普通微信用户，通过客服消息接口发送的消息称为“客服消息”。高级接口及“客服消息”将在第 5 章进行介绍。

4.1.1 消息交互流程

从用户发送交互请求，到公众号的接收与回应，中间会通过微信服务器的处理，并最终将返回的内容送至用户端，整个消息的交互过程如图 4-1 所示。

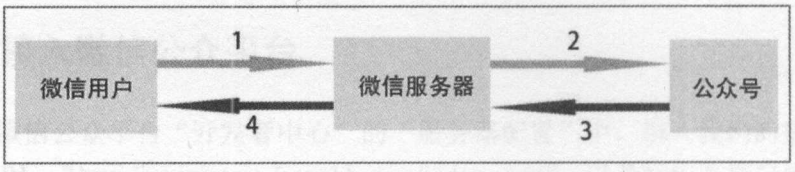


图 4-1

具体的消息交互过程如下：

1. 当普通微信用户向公众号发消息时，用户发送的消息首先会被发送到微信服务器上。

2. 微信服务器将用户消息封装为 XML 数据包，然后将封装后的消息通过 POST 请求方式发送到公众号填写的 URL 上。

3. 公众号服务器接收到用户消息后进行处理，处理完消息后，将需要回复的消息封装为微信公众平台指定的 XML 数据包，将 POST 请求结果返回给微信服务器。

4. 微信公众平台接收到 POST 请求结果后，再将公众号回复消息进行处理，并最终将回复内容返回给普通微信用户。

5. 整个消息交互过程为一次完整的 POST 请求。如果微信服务器在五秒内收不到 POST 请求的响应会断掉连接，并重新发起请求，总共重试三次。如重试三次仍未得到公众号服务器的回复，微信服务器将丢弃该普通用户发送的消息。

因此，我们的程序需要保证在五秒内回复 POST 请求给微信服务器。

4.1.2 消息数据结构

微信公众平台的消息是以 XML 数据格式进行封装传递的，消息分为用户发送消息及公众号回复消息。

以用户发送的文本消息为例，一个完整的消息的数据结构如下：

```
<xml>
<ToUserName><![CDATA[toUser]]></ToUserName>
<FromUserName><![CDATA[fromUser]]></FromUserName>
<CreateTime>1348831860</CreateTime>
<MsgType><![CDATA[text]]></MsgType>
<Content><![CDATA[this is a test]]></Content>
<MsgId>1234567890123456</MsgId>
</xml>
```

在 C#程序中处理消息时，需要将 XML 数据格式转换为 C#程序能识别的数据实体。

通过逐一分析每个消息的数据格式，发现所有的消息都有一些共同特点。所有的消息均含有如下基本信息：

◎ 消息接收方名称：ToUserName

◎ 消息发送方名称：FromUserName

◎ 消息创建时间：CreateTime

◎ 用户发送消息和公众号回复消息各有不同的消息类型：MsgType

◎ 用户发送消息，还含有消息 ID 信息：MsgId。

◎ 不同类型的消息还含有该类型消息特定的消息内容信息。

首先，我们根据消息的共同特性，定义一个所有的消息都需要遵循的数据格式规范 IMessageBase，这里利用 C#语言提供的特性“接口”来实现。同时，建立消息数据实体的基类 MessageBase。

```
/// <summary>
/// 所有 Request 和 Response 消息的数据格式规范
/// </summary>
public interface IMessageBase
{
    string ToUserName { get; set; }
    string FromUserName { get; set; }
    DateTime CreateTime { get; set; }
}
/// <summary>
/// 所有 Request 和 Response 消息的基类
/// </summary>
```

```
public class MessageBase
{
    public string ToUserName { get; set; }
    public string FromUserName { get; set; }
    public DateTime CreateTime { get; set; }
}
```

4.1.3 用户发送消息数据实体

根据“最新的微信公众平台开发者文档”，目前微信公众平台支持的用户发送消息类型有以下几种：

◎ 文本消息

◎ 图片消息

◎ 语音消息

◎ 视频消息

◎ 地理位置消息

◎ 链接消息

◎ 事件消息

据此，我们定义一个用户发送消息类型的枚举 RequestMsgType，以方便程序开发：

```
/// <summary>
/// 用户发送消息类型
/// </summary>
public enum RequestMsgType
{
    /// <summary>
    /// 文本消息
    /// </summary>
    Text,
    /// <summary>
    /// 地理位置消息
    /// </summary>
    Location,
```

```
        /// <summary>
        /// 图片消息
        /// </summary>
        Image,
        /// <summary>
        /// 语音消息
        /// </summary>
        Voice,
        /// <summary>
        /// 视频消息
        /// </summary>
        Video,
        /// <summary>
        /// 链接消息
        /// </summary>
        Link,
        /// <summary>
        /// 事件消息
        /// </summary>
        Event,
    }
```

接下来建立继承于所有消息基类 MessageBase 的用户发送消息数据实体的基类 RequestMessageBase。同时建立一个接口 IRequestMessageBase 来规范用户发送消息的数据格式，该接口同样需要遵循 IMessageBase 的数据格式规范：

```
    /// <summary>
    /// 用户发送消息数据格式规范
    /// </summary>
    public interface IRequestMessageBase : IMessageBase
    {
        /// <summary>
        /// 用户发送消息类型
        /// </summary>
        RequestMsgType MsgType { get; }
        /// <summary>
        /// 消息 ID
        /// </summary>
        long MsgId { get; set; }
    }
    /// <summary>
    /// 用户发送消息基类
```

```
    /// </summary>
    public class RequestMessageBase : MessageBase, IRequestMessageBase
    {
        /// <summary>
        /// 用户发送消息类型
        /// </summary>
        public virtual RequestMsgType MsgType
        {
            get { return RequestMsgType.Text; }
        }
        /// <summary>
        /// 消息 ID
        /// </summary>
        public long MsgId { get; set; }
    }
```

然后，我们建立每一种用户发送消息的数据实体，这些数据实体均继承于用户发送消息的基类 RequestMessageBase。

1. 文本消息

```
    public class RequestMessageText : RequestMessageBase,IRequestMessageBase
    {
        public override RequestMsgType MsgType
        {
            get { return RequestMsgType.Text; }
        }
        public string Content { get; set; }
    }
```

2. 图片消息

```
    public class RequestMessageImage : RequestMessageBase, IRequestMessageBase
    {
        public override RequestMsgType MsgType
        {
            get { return RequestMsgType.Image; }
        }
        public string MediaId { get; set; }
        public string PicUrl { get; set; }
    }
```

3. 语音消息

```
public class RequestMessageVoice : RequestMessageBase,IRequestMessageBase
{
    public override RequestMsgType MsgType
    {
        get { return RequestMsgType.Voice; }
    }
    public string MediaId { get; set; }
    /// <summary>
    /// 语音格式：amr
    /// </summary>
    public string Format { get; set; }
    /// <summary>
    /// 语音识别结果，UTF8 编码
    /// </summary>
    public string Recognition { get; set; }
}
```

4. 视频消息

```
public class RequestMessageVideo : RequestMessageBase, IRequestMessageBase
{
    public override RequestMsgType MsgType
    {
        get { return RequestMsgType.Video; }
    }
    public string MediaId { get; set;}
    public string ThumbMediaId { get; set; }
}
```

5. 地理位置消息

```
public class RequestMessageLocation : RequestMessageBase, IRequestMessageBase
{
    public override RequestMsgType MsgType
    {
        get { return RequestMsgType.Location; }
    }
    /// <summary>
    /// 地理位置纬度
```

```
        /// </summary>
        public double Location_X { get; set; }
        /// <summary>
        /// 地理位置经度
        /// </summary>
        public double Location_Y { get; set; }
        public int Scale { get; set; }
        public string Label { get; set; }
    }
```

6. 链接消息

```
public class RequestMessageLink : RequestMessageBase, IRequestMessageBase
{
    public override RequestMsgType MsgType
    {
        get { return RequestMsgType.Link; }
    }
    public string Title { get; set; }
    public string Description { get; set; }
    public string Url { get; set; }
}
```

4.1.4 用户发送事件消息数据实体

事件消息又分为以下几种不同的事件类型：

◎ 关注/取消关注事件

◎ 扫描带参数的二维码事件

◎ 上报地理位置事件

◎ 自定义菜单单击事件

同样定义一个事件类型的枚举 Event：

```
/// <summary>
/// 当RequestMsgType类型为Event时，Event属性的类型
/// </summary>
public enum Event
{
```

```
        /// <summary>
        /// 上报地理位置事件
        /// </summary>
        LOCATION,
        /// <summary>
        /// 关注事件
        /// </summary>
        subscribe,
        /// <summary>
        /// 关注事件
        /// </summary>
        unsubscribe,
        /// <summary>
        /// 自定义菜单单击事件
        /// </summary>
        CLICK,
        /// <summary>
        /// 扫描带参数二维码事件
        /// </summary>
        scan
    }
```

定义一个继承于用户发送消息基类 RequestMessageBase 的用户发送事件消息基类 RequestMessageEventBase，并创建事件接口 IRequestMessageEventBase 以规范数据格式。

```
    /// <summary>
    /// 用户发送事件消息接口
    /// </summary>
    public interface IRequestMessageEventBase : IRequestMessageBase
    {
        /// <summary>
        /// 事件类型
        /// </summary>
        Event Event { get; }
        /// <summary>
        /// 事件 KEY 值，与自定义菜单接口中的 KEY 值对应
        /// </summary>
        string EventKey { get; set; }
    }
    /// <summary>
    /// 用户发送事件消息基类
```

```
    /// </summary>
    public class RequestMessageEventBase : RequestMessageBase,
IRequestMessageEventBase
    {
        public override RequestMsgType MsgType
        {
            get { return RequestMsgType.Event; }
        }
        /// <summary>
        /// 事件类型
        /// </summary>
        public virtual Event Event
        {
            get { return Event.CLICK; }
        }
        /// <summary>
        /// 事件KEY值，与自定义菜单接口中的KEY值对应
        /// </summary>
        public string EventKey { get; set; }
    }
```

最后基于用户发送事件消息基类 RequestMessageEventBase，建立每一种事件类型的数据实体。

1. 关注事件

```
    public class RequestMessageEvent_Subscribe : RequestMessageEventBase,
IRequestMessageEventBase
    {
        public override Event Event
        {
            get { return Event.subscribe; }
        }
    }
```

2. 取消关注事件

```
    public class RequestMessageEvent_Unsubscribe : RequestMessageEventBase,
IRequestMessageEventBase
    {
        public override Event Event
```

```
        {
            get { return Event.unsubscribe; }
        }
    }
```

3. 扫描带参数的二维码事件

```
    public class RequestMessageEvent_Scan : RequestMessageEventBase,
IRequestMessageEventBase
    {
        public override Event Event
        {
            get { return Event.scan; }
        }
        /// <summary>
        /// 二维码的参数
        /// </summary>
        public string Ticket { get; set; }
    }
```

4. 上报地理位置事件

```
    public class RequestMessageEvent_Location : RequestMessageEventBase,
IRequestMessageEventBase
    {
        public override Event Event
        {
            get { return Event.LOCATION; }
        }
        /// <summary>
        /// 地理位置维度
        /// </summary>
        public double Latitude { get; set; }
        /// <summary>
        /// 地理位置经度
        /// </summary>
        public double Longitude { get; set; }
        /// <summary>
        /// 地理位置精度
        /// </summary>
```

```
        public double Precision { get; set; }
    }
```

5. 自定义菜单单击事件

```
    public class RequestMessageEvent_Click : RequestMessageEventBase,
IRequestMessageEventBase
    {
        public override Event Event
        {
            get { return Event.CLICK; }
        }
    }
```

4.1.5 公众号回复消息数据实体

目前微信公众平台支持的公众号回复消息类型有：

- ◎ 回复文本消息
- ◎ 回复图片消息
- ◎ 回复语音消息
- ◎ 回复视频消息
- ◎ 回复音乐消息
- ◎ 回复图文消息

首先定义一个公众号回复消息类型的枚举：

```
    /// <summary>
    /// 公众号回复消息类型
    /// </summary>
    public enum ResponseMsgType
    {
        /// <summary>
        /// 回复文本消息
        /// </summary>
        Text,
        /// <summary>
        /// 回复图文消息
```

```
    /// </summary>
    News,
    /// <summary>
    /// 回复音乐消息
    /// </summary>
    Music,
    /// <summary>
    /// 回复图片消息
    /// </summary>
    Image,
    /// <summary>
    /// 回复语音消息
    /// </summary>
    Voice,
    /// <summary>
    /// 回复视频消息
    /// </summary>
    Video
}
```

然后同样建立一个接口 IResponseMessageBase，一个基类 ResponseMessageBase：

```
/// <summary>
/// 公众号回复消息数据规范接口
/// </summary>
public interface IResponseMessageBase : IMessageBase
{
    /// <summary>
    /// 公众号回复消息类型
    /// </summary>
    ResponseMsgType MsgType { get; }
}
/// <summary>
/// 公众号回复消息基类
/// </summary>
public class ResponseMessageBase : MessageBase, IResponseMessageBase
{
    /// <summary>
    /// 公众号回复消息类型
    /// </summary>
    public virtual ResponseMsgType MsgType
    {
```

```
        get { return ResponseMsgType.Text; }
    }
}
```

接下来分别实现每种公众号回复消息的数据结构。

1. 回复文本消息

```
public class ResponseMessageText : ResponseMessageBase, IResponseMessageBase
{
    new public virtual ResponseMsgType MsgType
    {
        get { return ResponseMsgType.Text; }
    }
    public string Content { get; set; }
}
```

2. 回复图片消息

```
public class ResponseMessageImage : ResponseMessageBase, IResponseMessageBase
{
    public ResponseMessageImage()
    {
        Image = new Image();
    }
    new public virtual ResponseMsgType MsgType
    {
        get { return ResponseMsgType.Image; }
    }

    public Image Image { get; set; }
}
public class Image
{
    public string MediaId { get; set; }
}
```

3. 回复语音消息

```
public class ResponseMessageVoice : ResponseMessageBase, IResponseMessageBase
{
```

```
    public ResponseMessageVoice()
    {
        Voice = new Voice();
    }
    new public virtual ResponseMsgType MsgType
    {
        get { return ResponseMsgType.Voice; }
    }
    public Voice Voice { get; set; }
}
public class Voice
{
    public string MediaId { get; set; }
}
```

4. 回复视频消息

```
public class ResponseMessageVideo : ResponseMessageBase, IResponseMessageBase
{
    public ResponseMessageVideo()
    {
        Video = new Video();
    }
    new public virtual ResponseMsgType MsgType
    {
        get { return ResponseMsgType.Video; }
    }
    public Video Video { get; set; }
}
public class Video
{
    public string MediaId { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
}
```

5. 回复音乐消息

```
public class ResponseMessageMusic : ResponseMessageBase, IResponseMessageBase
{
    public ResponseMessageMusic()
```

```
        {
            Music = new Music();
        }
        public override ResponseMsgType MsgType
        {
            get { return ResponseMsgType.Music; }
        }
        public Music Music { get; set; }
    }
    public class Music
    {
        public string Title { get; set; }
        public string Description { get; set; }
        public string MusicUrl { get; set; }
        public string HQMusicUrl { get; set; }
        /// <summary>
        /// 缩略图的媒体 ID，通过上传多媒体文件，得到的 ID
        /// </summary>
        //public string ThumbMediaId { get; set; }
    }
```

6. 回复图文消息

```
public class ResponseMessageNews : ResponseMessageBase, IResponseMessageBase
{
    public ResponseMessageNews()
    {
        Articles = new List<Article>();
    }
    new public virtual ResponseMsgType MsgType
    {
        get { return ResponseMsgType.News; }
    }
    public int ArticleCount
    {
        get { return (Articles ?? new List<Article>()).Count; }
        set
        {
        //这里开放 set 只为了逆向从 Response 的 XML 转成实体的操作一致性，没有实际意义
        }
```

```
    }
    /// <summary>
    /// 文章列表，微信客户端只能输出前 10 条（可能未来数字会有变化，出于视觉效果考虑，
    /// 建议控制在 8 条以内）
    /// </summary>
    public List<Article> Articles { get; set; }
}
public class Article
{
    public string Title { get; set; }
    public string Description { get; set; }
    public string PicUrl { get; set; }
    public string Url { get; set; }
}
```

4.1.6　消息数据转换

在完成所有微信消息对应的 C#数据实体的编写工作后，我们需要开发将 XML 数据包转换为对应 C#数据实体的方法，用于在接收到微信服务器传递的用户发送消息后，将消息转换为 C#能处理的数据实体，以及将 C#数据实体转换为 XML 数据包的方法，用于将公众号返回消息从 C#数据实体转换为微信服务器所需的 XML 数据包。

由于 HTTP 协议传递的是字符串形式的数据，我们首先需要将字符串转换为 C#能处理的 XML 数据实体，.NET Framework 已经提供了一个 XML 数据实体与字符串互转，以及操作 XML 数据的一整套完整方法。我们只需要在程序中引用命名空间 System.Xml.Linq，然后调用 XDocument 类即可。

XDocument.Parse 方法实现将字符串转换为 XML 数据实体，XDocument.Tostring 方法实现将 XML 实体转换为字符串，这两个函数的定义如下：

```
// 摘要
//     从字符串创建新 System.Xml.Linq.XDocument
// 参数
//   text:
//     包含 XML 的字符串
// 返回结果
//     一个使用包含 XML 的字符串填充的 System.Xml.Linq.XDocument
public static XDocument Parse(string text);
```

```
    // 摘要
    //     返回此节点的缩进 XML
    // 返回结果
    //     一个包含缩进 XML 的 System.String
    public override string ToString();
```

为了便于识别消息处理过程中产生的错误，我们首先对消息处理过程中可能产生的错误进行定义。错误分为两类，一类是我们在开发过程中预先考虑到可能产生的错误，这种错误类型定义为 WeixinException。另一类是我们没有考虑到的未知错误，定义为 UnknownRequestMsgTypeException。它们都继承于.NET Framework 的错误类型 ApplicationException。

```
/// <summary>
/// 微信自定义异常基类
/// </summary>
public class WeixinException : ApplicationException
{
    public WeixinException(string message)
        : base(message, null)
    {}
    public WeixinException(string message, Exception inner)
        : base(message, inner)
    {}
}
/// <summary>
/// 未知请求类型
/// </summary>
public class UnknownRequestMsgTypeException : WeixinException
{
    public UnknownRequestMsgTypeException(string message)
        : base(message, null)
    {}
    public UnknownRequestMsgTypeException(string message, Exception inner)
        : base(message, inner)
    { }
}
```

1. 用户发送消息处理

在接收到微信服务器发送的用户发送消息后，首先需要判断消息的类型，根据

具体的消息类型实例化一个该消息类型的数据实体，我们定义一个 GetRequestEntity 方法，使用 switch-case 语法来进行判断，并定义两个工具类 MsgTypeHelper 和 EventHelper，来实现获取 XML 数据包的消息类型和事件类型。如果发现消息类型不在已定义的数据类型中，抛出 UnknownRequestMsgTypeException 错误，以便进行调试。在实际的项目中，所有发生的错误都应该记录到错误日志中，方便项目正式运行后发现及修复 BUG。

```
/// <summary>
/// 获取 XDocument 转换后的 IRequestMessageBase 实例
/// 如果 MsgType 不存在，则抛出 UnknownRequestMsgTypeException 异常
/// </summary>
/// <returns></returns>
public static IRequestMessageBase GetRequestEntity(XDocument doc)
{
    RequestMessageBase requestMessage = null;
    RequestMsgType msgType;
    try
    {
        msgType = MsgTypeHelper.GetRequestMsgType(doc);
        switch (msgType)
        {
            case RequestMsgType.Text:
                requestMessage = new RequestMessageText();
                break;
            case RequestMsgType.Location:
                requestMessage = new RequestMessageLocation();
                break;
            case RequestMsgType.Image:
                requestMessage = new RequestMessageImage();
                break;
            case RequestMsgType.Voice:
                requestMessage = new RequestMessageVoice();
                break;
            case RequestMsgType.Video:
                requestMessage = new RequestMessageVideo();
                break;
            case RequestMsgType.Link:
                requestMessage = new RequestMessageLink();
                break;
            case RequestMsgType.Event:
```

```
                Event eventType = EventHelper.GetEventType(doc);
                //判断 Event 类型
                switch (eventType)
                {
                    case Event.LOCATION:
                        requestMessage = new RequestMessageEvent_Location();
                        break;
                    case Event.subscribe:
                        requestMessage = new RequestMessageEvent_Subscribe();
                        break;
                    case Event.unsubscribe:
                     requestMessage = new RequestMessageEvent_Unsubscribe();
                        break;
                    case Event.CLICK:
                        requestMessage = new RequestMessageEvent_Click();
                        break;
                    case Event.scan:
                        requestMessage = new RequestMessageEvent_Scan();
                        break;
                    default://其他意外类型（也可以选择抛出异常）
                        requestMessage = new RequestMessageEventBase();
                        break;
                }
                break;
            default:
              throw new UnknownRequestMsgTypeException(string.Format("MsgType:
{0} 在 RequestMessageFactory 中没有对应的处理程序！", msgType), new
ArgumentOutOfRangeException());//为了能够对类型变动最大程度地容错(如微信目前还可以对
//公众账号 suscribe 等未知类型，但 API 没有开放)，建议在使用的时候 catch 这个异常
            }
             //根据 XML 信息填充实体
            FillEntityWithXml(requestMessage, doc);
        }
        catch (ArgumentException ex)
        {
            throw new WeixinException(string.Format("RequestMessage 转换出错！可
能是 MsgType 不存在!, XML: {0}", doc.ToString()), ex);
        }
        return requestMessage;
    }
    public static class MsgTypeHelper
```

```
    {
        #region RequestMsgType
        /// <summary>
        /// 根据 xml 信息，返回 RequestMsgType
        /// </summary>
        /// <returns></returns>
        public static RequestMsgType GetRequestMsgType(XDocument doc)
        {
            return GetRequestMsgType(doc.Root.Element("MsgType").Value);
        }
        /// <summary>
        /// 根据 XML 信息，返回 RequestMsgType
        /// </summary>
        /// <returns></returns>
        public static RequestMsgType GetRequestMsgType(string str)
        {
            return  (RequestMsgType)Enum.Parse(typeof(RequestMsgType),  str,
true);
        }
        #endregion
    }
    public class EventHelper
    {
        /// <summary>
        /// 根据 XML 信息，返回 EventType
        /// </summary>
        public static Event GetEventType(XDocument doc)
        {
            return GetEventType(doc.Root.Element("Event").Value);
        }
        public static Event GetEventType(string str)
        {
            return (Event)Enum.Parse(typeof(Event), str, true);
        }
    }
```

在识别出消息类型后，将 XML 数据包中的具体信息逐一填充到该类型消息的数据实体中。C#具有的“泛型”特性，这样我们就不需要针对每种消息类型单独编写填充方法，所有消息类型的填充方法可以写在一个方法 FillEntityWithXml 中，通过 switch-case 语法实现不同类型消息的处理。

具体的填充算法使用C#提供的特性“反射”来递归实现，思路如下：

（1）获取数据实体中每个数据字段的信息：entity.GetType().GetProperties()。

（2）对某一数据字段判断 XML 数据包中是否有相同名称的数据信息，如果有，则将 XML 数据包中该数据字段的值填入数据实体的对应数据字段中。

（3）对含有多个数据的数据字段，递归调用 FillEntityWithXml，解析每个子数据字段的信息并进行填充。

（4）重复执行（2）、（3）步骤，直到所有的数据字段都被填充完毕。

```
    /// <summary>
    /// 根据 XML 信息填充实体
    /// </summary>
    /// <typeparam name="T">MessageBase 为基类的类型，Response 和 Request 都可以
</typeparam>
    /// <param name="entity">实体</param>
    /// <param name="doc">XML</param>
    public static void FillEntityWithXml<T>(this T entity, XDocument doc) where
T : /*MessageBase*/ class, new()
    {
        entity = entity ?? new T();
        var root = doc.Root;
        var props = entity.GetType().GetProperties();
        foreach (var prop in props)
        {
            var propName = prop.Name;
            if (root.Element(propName) != null)
            {
                switch (prop.PropertyType.Name)
                {
                    case "DateTime":
                        prop.SetValue(entity, DateTimeHelper.GetDateTimeFromXml
(root.Element(propName).Value), null);
                        break;
                    case "Int32":
                        prop.SetValue(entity, int.Parse(root.Element(propName).
Value), null);
                        break;
                    case "Int64":
                        prop.SetValue(entity, long.Parse(root.Element(propName).
```

```
Value), null);
                    break;
                case "Double":
                    prop.SetValue(entity, double.Parse(root.Element
(propName).Value), null);
                    break;
                //以下为枚举类型
                case "RequestMsgType":
                case "ResponseMsgType":
                case "Event":
                    break;
                //以下为实体类型
                case "List`1"://List<T>类型，ResponseMessageNews 适用
                 var genericArguments = prop.PropertyType.GetGenericArguments();
                     if (genericArguments[0].Name == "Article")
                     //ResponseMessageNews 适用
                    {
                        //文章下属节点 item
                        List<Article> articles = new List<Article>();
                        foreach (var item in root.Element(propName).Elements
("item"))
                        {
                            var article = new Article();
                            FillEntityWithXml(article, new XDocument(item));
                            articles.Add(article);
                        }
                        prop.SetValue(entity, articles, null);
                    }
                    break;
                case "Music"://ResponseMessageMusic 适用
                    Music music = new Music();
                    FillEntityWithXml(music, new XDocument(root.Element
(propName)));
                    prop.SetValue(entity, music, null);
                    break;
                case "Image"://ResponseMessageImage 适用
                    Image image = new Image();
                    FillEntityWithXml(image,   new   XDocument(root.Element
(propName)));
                    prop.SetValue(entity, image, null);
                    break;
```

```
                case "Voice"://ResponseMessageVoice 适用
                    Voice voice = new Voice();
                    FillEntityWithXml(voice, new XDocument(root.Element
(propName)));
                    prop.SetValue(entity, voice, null);
                    break;
                case "Video"://ResponseMessageVideo 适用
                    Video video = new Video();
                    FillEntityWithXml(video, new XDocument(root.Element
(propName)));
                    prop.SetValue(entity, video, null);
                    break;
                default:
                   prop.SetValue(entity, root.Element(propName).Value, null);
                    break;
            }
        }
    }
}
public static class DateTimeHelper
{
    public static DateTime BaseTime = new DateTime(1970, 1, 1);
    //UNIX 起始时间
    /// <summary>
    /// 转换微信 DateTime 时间到 C#时间
    /// </summary>
    /// <param name="dateTimeFromXml">微信 DateTime</param>
    /// <returns></returns>
    public static DateTime GetDateTimeFromXml(long dateTimeFromXml)
    {
        return BaseTime.AddTicks((dateTimeFromXml + 8 * 60 * 60) * 10000000);
    }
    /// <summary>
    /// 转换微信 DateTime 时间到 C#时间
    /// </summary>
    /// <param name="dateTimeFromXml">微信 DateTime</param>
    /// <returns></returns>
    public static DateTime GetDateTimeFromXml(string dateTimeFromXml)
    {
        return GetDateTimeFromXml(long.Parse(dateTimeFromXml));
    }
```

```
    /// <summary>
    /// 获取微信 DateTime
    /// </summary>
    /// <param name="dateTime">时间</param>
    /// <returns></returns>
    public static long GetWeixinDateTime(DateTime dateTime)
    {
        return (dateTime.Ticks - BaseTime.Ticks) / 10000000 - 8 * 60 * 60;
    }
}
```

由于 XML 数据包中的 CreateTime 字段保存的时间信息是 UNIX 时间戳格式，该格式将时间表示为从 1970 年 1 月 1 日到当前时间的秒数。我们需要建立一个时间转换类 DateTimeHelper，进行 UNIX 时间戳格式和 C#程序 DateTime 格式的转换。

```
public static class DateTimeHelper
{
    public static DateTime BaseTime = new DateTime(1970, 1, 1);
    // UNIX 起始时间
    /// <summary>
    /// 转换微信 DateTime 时间到 C#时间
    /// </summary>
    /// <param name="dateTimeFromXml">微信 DateTime</param>
    /// <returns></returns>
    public static DateTime GetDateTimeFromXml(long dateTimeFromXml)
    {
        return BaseTime.AddTicks((dateTimeFromXml + 8 * 60 * 60) * 10000000);
    }
    /// <summary>
    /// 转换微信 DateTime 时间到 C#时间
    /// </summary>
    /// <param name="dateTimeFromXml">微信 DateTime</param>
    /// <returns></returns>
    public static DateTime GetDateTimeFromXml(string dateTimeFromXml)
    {
        return GetDateTimeFromXml(long.Parse(dateTimeFromXml));
    }
    /// <summary>
    /// 获取微信 DateTime
    /// </summary>
    /// <param name="dateTime">时间</param>
    /// <returns></returns>
```

```
        public static long GetWeixinDateTime(DateTime dateTime)
        {
            return (dateTime.Ticks - BaseTime.Ticks) / 10000000 - 8 * 60 * 60;
        }
    }
```

2. 公众号回复消息处理

公众号回复消息的消息处理过程与用户发送信息相反，定义一个ConvertEntityToXml方法来实现数据实体到XML数据包的转换。其算法思路与用户发送信息的消息处理过程类似，同样使用C#的特性“泛型”与“反射”。

（1）新建一个空的XML数据包。

（2）将数据实体中数据字段的名称与值填写到XML数据包中。

（3）对含有多个数据的数据字段，递归调用ConvertEntityToXml方法，将每个子数据字段填充到XML数据包中。

（4）重复执行（2）、（3）步骤，直到所有的数据字段都被填充到XML数据实体中。

```
    /// <summary>
    /// 将实体转为 XML
    /// </summary>
    /// <typeparam name="T">RequestMessage 或 ResponseMessage</typeparam>
    /// <param name="entity">实体</param>
    /// <returns>XML</returns>
    public static XDocument ConvertEntityToXml<T>(this T entity) where T :
class , new()
    {
        entity = entity ?? new T();
        var doc = new XDocument();
        doc.Add(new XElement("xml"));
        var root = doc.Root;
        /* 注意!
            * 经过测试，微信对字段排序有严格要求，这里对排序进行强制约束
        */
        var propNameOrder = new List<string>() { "ToUserName", "FromUserName",
"CreateTime", "MsgType" };
        //不同的返回类型需要对应不同的特殊格式进行排序
```

```
        if (entity is ResponseMessageNews)
        {
            propNameOrder.AddRange(new[] { "ArticleCount", "Articles",/*以下是
Atricle属性*/ "Title ", "Description ", "PicUrl", "Url" });
        }
        else if (entity is ResponseMessageMusic)
        {
            propNameOrder.AddRange(new[] { "Music",/*以下是Music属性*/ "Title ",
"Description ", "MusicUrl", "HQMusicUrl" });
        }
        else if (entity is ResponseMessageImage)
        {
            propNameOrder.AddRange(new[] { "Image",/*以下是Image属性*/ "MediaId " });
        }
        else if (entity is ResponseMessageVoice)
        {
            propNameOrder.AddRange(new[] { "Voice",/*以下是Voice属性*/ "MediaId " });
        }
        else if (entity is ResponseMessageVideo)
        {
            propNameOrder.AddRange(new[] { "Video",/*以下是Video属性*/ "MediaId
", "Title", "Description" });
        }
        else
        {
            //如Text类型
            propNameOrder.AddRange(new[] { "Content" });
        }
        Func<string, int> orderByPropName = propNameOrder.IndexOf;
        var props = entity.GetType().GetProperties().OrderBy(p => orderByPropName
(p.Name)).ToList();
        foreach (var prop in props)
        {
            var propName = prop.Name;
            if (propName == "Articles")
            {
                //文章列表
                var atriclesElement = new XElement("Articles");
                var articales = prop.GetValue(entity, null) as List<Article>;
                foreach (var articale in articales)
                {
```

```
                var subNodes = ConvertEntityToXml(articale).Root.Elements();
                atriclesElement.Add(new XElement("item", subNodes));
            }
            root.Add(atriclesElement);
        }
        else if (propName == "Music" || propName == "Image" || propName ==
"Video" || propName == "Voice")
        {
            //音乐、图片、视频、语音格式
            var musicElement = new XElement(propName);
            var media = prop.GetValue(entity, null);// as Music;
            var subNodes = ConvertEntityToXml(media).Root.Elements();
            musicElement.Add(subNodes);
            root.Add(musicElement);
        }
        else
        {
            switch (prop.PropertyType.Name)
            {
                case "String":
                    root.Add(new XElement(propName, new XCData(prop.GetValue
(entity, null) as string ?? "")));
                    break;
                case "DateTime":
                    root.Add(new XElement(propName, DateTimeHelper.
GetWeixinDateTime((DateTime)prop.GetValue(entity, null))));
                    break;
                case "ResponseMsgType":
                    root.Add(new XElement(propName, new XCData(prop.GetValue
(entity,null).ToString().ToLower())));
                    break;
                case "Article":
                    root.Add(new XElement(propName, prop.GetValue(entity, null).
ToString().ToLower()));
                    break;
                default:
                    root.Add(new XElement(propName, prop.GetValue(entity, null)));
                    break;
            }
        }
    }
```

```
        return doc;
    }
```

4.2　用户会话上下文框架

由于微信公众平台的特殊机制，所有的信息都由微信服务器转发而来。这种方式有一个先天的缺陷：不同用户的请求都来自同一个微信服务器。Web 服务器一般是使用 Session 机制来识别不同的用户请求，并通过 Session 来保存每个用户的独立信息，而在微信公众平台的应用场景下，公众号服务器始终面对的是同一个请求对象，这样就导致公众号服务器无法识别不同用户的请求，无法使用 Session 对用户会话的上下文进行管理。

4.2.1　用户会话上下文应用场景

我们假定一个这样的应用场景，微信公众号要实现一个帮助普通微信用户寻找附近商店的功能，具体需求如下：

1．普通微信用户发送一个自己的地理位置消息给公众号。

2．微信公众号回复一条消息，告诉普通微信用户地理位置已收到，并请普通微信用户告诉微信公众号需要查找附近哪种类型的商店。

3．普通微信用户发送一个商店类型的文字消息，如“化妆品”等。

4．微信公众号收到消息，并查找到用户附近的化妆品商店后，回复一条多图文消息给普通微信用户。

由于微信的功能限制，普通微信用户每次只能发送一种类型的消息。想要获取附近商店的信息，普通微信用户必须分两次发送一条地理位置消息和一条文本消息。要实现这样的需求，微信公众号需要先保存第一条地理位置消息，并在普通微信用户发送第二条文本消息后，读取出前一条地理位置消息才能进行查找附近商店的程序处理。这就要求微信公众号能识别不同的普通微信用户，并对每个普通微信用户的消息按顺序进行独立保存。

微信公众平台的消息交互方式决定了无法使用 Session 对用户会话的上下文进行管理，我们需要单独设计一套适用于微信公众平台消息交互的用户会话上下

文机制。

4.2.2 用户会话上下文结构

微信公众号用户会话上下文需要满足以下几个条件：

1. 有一个集合来保存不同用户的会话上下文。
2. 每个用户一个独立的会话上下文。
3. 每个用户的会话上下文通过用户名来进行区别。
4. 每个用户的会话上下文能按顺序保存用户发送的消息记录。
5. 每个用户的会话上下文能按顺序保存用户接收的消息记录。
6. 为了保证程序执行效率，整个用户会话上下文集合保存在内存中。

一个微信公众号保存的用户会话上下文结构如图 4-2 所示。

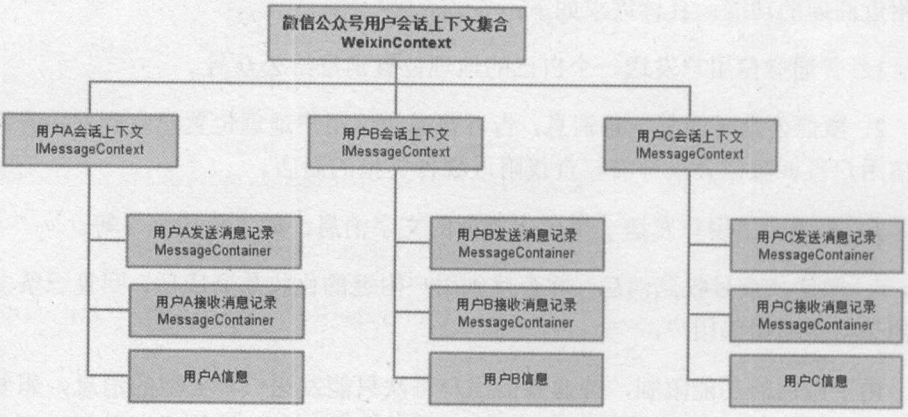


图 4-2

4.2.3 发送与接收消息记录

由于每条发送的消息记录和接收的消息记录都会占用一定的内存，为了避免保存的消息记录过多，导致占用内存太大而使程序崩溃。我们需要限制每个用户保存的消息记录条数，并且在消息记录超过限制后，移除最早的消息。

用户的消息记录通过建立一个继承于 System.Collections.Generic.List 集合类的 MessageContainer 集合类来进行保存。该类定义了消息最大保存数量 MaxRecordCount，在记录末尾添加消息记录的方法 Add、AddRange，以及插入消息记录的方法 Insert、InsertRange。在添加或插入消息记录的时候，调用 RemoveExpressItems 来移除超过数量限制的最早消息。

```
public class MessageContainer<T> : List<T>
{
    /// <summary>
    /// 最大记录条数（保留尾部），如果小于或等于 0 则不限制
    /// </summary>
    public int MaxRecordCount { get; set; }
    private MessageContainer()
    {
    }
    public MessageContainer(int maxRecordCount)
    {
        MaxRecordCount = maxRecordCount;
    }
    new public void Add(T item)
    {
        base.Add(item);
        RemoveExpressItems();
    }
    new public void AddRange(IEnumerable<T> collection)
    {
        base.AddRange(collection);
        RemoveExpressItems();
    }
    new public void Insert(int index, T item)
    {
        base.Insert(index, item);
        RemoveExpressItems();
    }
    new public void InsertRange(int index, IEnumerable<T> collection)
    {
        base.InsertRange(index, collection);
        RemoveExpressItems();
    }
    /// <summary>
    /// 如果消息记录超过数量限制，移除最早的消息
```

```
        /// </summary>
        private void RemoveExpressItems()
        {
            if (MaxRecordCount > 0 && base.Count > MaxRecordCount)
            {
                base.RemoveRange(0, base.Count - MaxRecordCount);
            }
        }
    }
```

4.2.4 用户会话上下文信息

接下来定义一个接口 IMessageContext 用于规定单个用户会话上下文需要保存的信息。用户会话上下文的基本信息包括：

1. 微信公众平台用来唯一标识一个普通微信用户的 OpenID。

2. 用户发送消息记录 RequestMessages。

3. 用户接收消息记录 ResponseMessages。

4. 消息的最大存储数量 MaxRecordCount。

由于目前设计的框架均与具体的业务无关，为了能适应具体的业务场景，考虑到框架的可扩展性，增加一个用于存储任何和用户上下文有关数据的容器 StorageData，该容器在框架中没有被使用到，完全由具体的业务场景决定里面的内容（比如用户执行到哪一步、或某个比较重要的位置信息等），类似于 Session 的作用。

如果向一个微信公众号发送消息的用户数量太多，那么会导致微信公众号用户会话上下文集合中保存的用户会话上下文数量过多，占用内存空间过大，导致程序崩溃。因此需要设计一个机制来限制微信公众号用户会话上下文集合中保存的用户会话上下文数量。

参考 Session 机制超时时间的概念，限制微信公众号用户会话上下文集合数量的机制为：将长期没有活动即用户会话上下文长期未更新的用户移除集合。

为了实现超时移除集合机制，用户会话上下文需要增加以下内容：

1. 一个用来记录用户最后发送消息时间的变量 LastActiveTime，用于判断用

户是否长期未活动，是否需要从集合中移除该用户会话上下文。

2．一个供集合移除用户会话上下文时调用的回调方法 OnRemoved。

3．一个用户会话上下文被从集合中移除时触发的事件 MessageContextRemoved，由具体的业务场景根据需求，编写用户上下文被从集合移除时需要额外执行的方法注册到该事件上。

```
public interface IMessageContext
{
    /// <summary>
    /// 用户名（OpenID）
    /// </summary>
    string UserName { get; set; }
    /// <summary>
    /// 最后一次活动时间（用户主动发送 Resquest 请求的时间）
    /// </summary>
    DateTime LastActiveTime { get; set; }
    /// <summary>
    /// 用户发送消息记录
    /// </summary>
    MessageContainer<IRequestMessageBase> RequestMessages { get; set; }
    /// <summary>
    /// 用户接收消息记录
    /// </summary>
    MessageContainer<IResponseMessageBase> ResponseMessages { get; set; }
    /// <summary>
    /// 最大存储容量（分别针对 RequestMessages 和 ResponseMessages）
    /// </summary>
    int MaxRecordCount { get; set; }
    /// <summary>
    /// 临时存储数据，如用户状态等，出于保持.net 3.5 版本，这里暂不使用 dynamic
    /// </summary>
    object StorageData { get; set; }
    /// <summary>
    /// 由具体的业务场景根据需求，编写用户上下文被从集合移除时需要额外执行的方法注册到
该事件上
    /// </summary>
    event EventHandler<WeixinContextRemovedEventArgs> MessageContextRemoved;
    /// <summary>
    /// 供集合移除用户会话上下文时调用的回调方法
    /// </summary>
```

```
        void OnRemoved();
    }
```

接下来编写一个实现接口 IMessageContext 的类 MessageContext，用于存储用户上下文信息。

```
    /// <summary>
    /// 微信用户上下文信息（单个用户）
    /// </summary>
    public class MessageContext : IMessageContext
    {
        private int _maxRecordCount;
        public string UserName { get; set; }
        public DateTime LastActiveTime { get; set; }
        public MessageContainer<IRequestMessageBase> RequestMessages { get;
set; }
        public MessageContainer<IResponseMessageBase> ResponseMessages { get;
set; }
        public int MaxRecordCount
        {
            get
            {
                return _maxRecordCount;
            }
            set
            {
                RequestMessages.MaxRecordCount = value;
                ResponseMessages.MaxRecordCount = value;

                _maxRecordCount = value;
            }
        }
        public object StorageData { get; set; }
        public event EventHandler<WeixinContextRemovedEventArgs>
MessageContextRemoved = null;
        /// <summary>
        /// 构造函数
        /// </summary>
        /// <param name="maxRecordCount">maxRecordCount 如果小于等于 0，则不限制
</param>
        public MessageContext()
        {
```

```
            /*
                * 注意：即使使用其他类实现 IMessageContext，
                * 也务必在这里进行下面的初始化，尤其是设置当前时间，
                * 这个时间关系到及时从缓存中移除过期的消息，节约内存使用
                */
            RequestMessages = new MessageContainer<IRequestMessageBase>
(MaxRecordCount);
            ResponseMessages = new MessageContainer<IResponseMessageBase>
            (MaxRecordCount);
            LastActiveTime = DateTime.Now;
        }
        public virtual void OnRemoved()
        {
            var onRemovedArg = new WeixinContextRemovedEventArgs(this);
            OnMessageContextRemoved(onRemovedArg);
        }
        /// <summary>
        /// 执行上下文被移除的事件
        /// 注意：此事件不是实时触发的，而是等过期后，在任意一个人发过来的下一条消息执行之
前触发
        /// </summary>
        /// <param name="e"></param>
        private void OnMessageContextRemoved(WeixinContextRemovedEventArgs e)
        {
          EventHandler<WeixinContextRemovedEventArgs> temp = MessageContextRemoved;
            if (temp != null)
            {
                temp(this, e);
            }
        }
    }
```

4.2.5　用户会话上下文集合

最后，定义一个保存微信公众号中所有用户会话上下文的集合类 WeixinContext。

在微信公众平台中，每个普通微信用户有一个唯一标识 OpenID，OpenID 会在消息中被传递。我们采用一种以 OpenID 为键，MessageContext 为值的字典数据结构“Dictionary<string, IMessageContext>MessageCollection”来保存用户会话

上下文集合。Dictionary 能通过键直接获取值，可以快速获取不同用户的用户会话上下文，避免查询产生性能损耗。

为了能快速的移除长期未活动的用户会话上下文，将每个用户会话上下文按最后一次活动时间升序排列，保存到一个“List<IMessageContext> MessageQueue”中。这样，每次执行移除长期未活动的用户会话上下文操作，只需要判断 MessageQueue 中前几个用户会话上下文是否应该移除即可，避免了每次移除都需要判断所有用户会话上下文，以提高程序执行效率。

在代码中，我们定义了一个开关 UseWeixinContext，来决定是否开启用户会话上下文。如果一些业务场景不需要用户会话上下文，可以关闭用户会话上下文功能，以节约资源，提高程序效率。

```
public static class WeixinContextGlobal
{
    public static object Lock = new object();
    /// <summary>
    /// 是否开启上下文记录
    /// </summary>
    public static bool UseWeixinContext = true;
}
/// <summary>
/// 对话上下文被删除时触发事件的事件数据
/// </summary>
public class WeixinContextRemovedEventArgs: EventArgs
{
    /// <summary>
    /// 该用户的 OpenId
    /// </summary>
    public string OpenId
    {
        get
        {
            return MessageContext.UserName;
        }
    }
    /// <summary>
    /// 最后一次响应时间
    /// </summary>
    public DateTime LastActiveTime
```

```
        {
            get
            {
                return MessageContext.LastActiveTime;
            }
        }
        /// <summary>
        /// 上下文对象
        /// </summary>
        public IMessageContext MessageContext { get; set; }
        public WeixinContextRemovedEventArgs(IMessageContext messageContext)
        {
            MessageContext = messageContext;
        }
    }
    /// <summary>
    /// 微信消息上下文（全局）
    /// 默认过期时间：90 分钟
    /// </summary>
    public class WeixinContext<TM> where TM : class, IMessageContext, new()
    {
        /// <summary>
        /// 默认过期时间
        /// </summary>
        private const int DEFAULTEXPIREMINUTES = 90;
        /// <summary>
        /// 所有 MessageContext 集合
        /// </summary>
        protected Dictionary<string, TM> MessageCollection { get; set; }
        /// <summary>
        /// MessageContext 列队（LastActiveTime 升序排列）
        /// </summary>
        protected List<TM> MessageQueue { get; set; }
        /// <summary>
        /// 每一个 MessageContext 过期时间
        /// </summary>
        public Double ExpireMinutes { get; set; }
        /// <summary>
        /// 最大存储上下文数量（分别针对请求和响应信息）
        /// </summary>
        public int MaxRecordCount { get; set; }
```

```
        public WeixinContext()
        {
            Restore();
        }
        /// <summary>
        /// 重置所有上下文参数，所有记录将被清空
        /// </summary>
        public void Restore()
        {
            MessageCollection = new Dictionary<string, TM>(StringComparer.
OrdinalIgnoreCase);
            MessageQueue = new List<TM>();
            ExpireMinutes = DEFAULTEXPIREMINUTES;
        }
        /// <summary>
        /// 获取 MessageContext，如果不存在，则返回 null
        /// 这个方法的更重要意义在于操作 TM 队列，及时移除过期信息，并将最新活动的对象移到
尾部
        /// </summary>
        /// <param name="userName">用户名（OpenId）</param>
        /// <returns></returns>
        private TM GetMessageContext(string userName)
        {
            //检查并移除过期记录，为了尽量节约资源，这里暂不使用独立线程轮询
            while (MessageQueue.Count > 0)
            {
                var firstMessageContext = MessageQueue[0];
                var timeSpan = DateTime.Now - firstMessageContext.LastActiveTime;
                if (timeSpan.TotalMinutes >= ExpireMinutes)
                {
                    MessageQueue.RemoveAt(0);//从队列中移除过期对象
                    MessageCollection.Remove(firstMessageContext.UserName);//
从集合中删除过期对象
                    //添加事件回调
                    firstMessageContext.OnRemoved();//TODO:此处异步处理，或用户在自
己操作的时候异步处理需要耗费时间比较长的操作
                }
                else
                {
                    break;
                }
```

```
            }
            /*
                * 全局只有在这里用到MessageCollection.ContainsKey,
                * 充分分离MessageCollection内部操作,
                * 为以后变化或扩展MessageCollection留余地
                */
            if (!MessageCollection.ContainsKey(userName))
            {
                return null;
            }
            return MessageCollection[userName];
        }
        /// <summary>
        /// 获取MessageContext
        /// </summary>
        /// <param name="userName">用户名（OpenId）</param>
        /// <param name="createIfNotExists">True：如果用户不存在，则创建一个实例，
并返回这个最新的实例
        /// False：如果用户存在，则返回null</param>
        /// <returns></returns>
        private TM GetMessageContext(string userName, bool createIfNotExists)
        {
            var messageContext = GetMessageContext(userName);

            if (messageContext == null)
            {
                if (createIfNotExists)
                {
                    //全局只在这一个地方使用MessageCollection[Key]写入
                    MessageCollection[userName] = new TM()
                    {
                        UserName = userName,
                        MaxRecordCount = MaxRecordCount
                    };
                    messageContext = GetMessageContext(userName);
                    //插入列队
                    MessageQueue.Add(messageContext); //最新的排到末尾
                }
                else
                {
                    return null;
                }
```

```
            }
            return messageContext;
        }
        /// <summary>
        /// 获取MessageContext，如果不存在，则使用requestMessage信息初始化一个，并
返回原始实例
        /// </summary>
        /// <returns></returns>
        public TM GetMessageContext(IRequestMessageBase requestMessage)
        {
            lock (WeixinContextGlobal.Lock)
            {
                return GetMessageContext(requestMessage.FromUserName, true);
            }
        }
        /// <summary>
        /// 获取MessageContext，如果不存在，则使用requestMessage信息初始化一个，并
返回原始实例
        /// </summary>
        /// <returns></returns>
        public TM GetMessageContext(IResponseMessageBase responseMessage)
        {
            lock (WeixinContextGlobal.Lock)
            {
                return GetMessageContext(responseMessage.ToUserName, true);
            }
        }
        /// <summary>
        /// 记录请求信息
        /// </summary>
        /// <param name="requestMessage">请求信息</param>
        public void InsertMessage(IRequestMessageBase requestMessage)
        {
            lock (WeixinContextGlobal.Lock)
            {
                var userName = requestMessage.FromUserName;
                var messageContext = GetMessageContext(userName, true);
                if (messageContext.RequestMessages.Count > 0)
                {
                    //如果不是新建对象，则把当前对象移到队列尾部（新对象已经在底部）
                    var messageContextInQueue =
                        MessageQueue.FindIndex(z => z.UserName == userName);
```

```
                if (messageContextInQueue >= 0)
                {
                    MessageQueue.RemoveAt(messageContextInQueue); //移除当前对象
                    MessageQueue.Add(messageContext); //插入到末尾
                }
            }
            messageContext.LastActiveTime = DateTime.Now;//记录请求时间
            messageContext.RequestMessages.Add(requestMessage);//录入消息
        }
    }
    /// <summary>
    /// 记录响应信息
    /// </summary>
    /// <param name="responseMessage">响应信息</param>
    public void InsertMessage(IResponseMessageBase responseMessage)
    {
        lock (WeixinContextGlobal.Lock)
        {
            var messageContext = GetMessageContext(responseMessage.ToUserName,
true);
            messageContext.ResponseMessages.Add(responseMessage);
        }
    }
    /// <summary>
    /// 获取最新一条请求数据，如果不存在，则返回 null
    /// </summary>
    /// <param name="userName">用户名（OpenId）</param>
    /// <returns></returns>
    public IRequestMessageBase GetLastRequestMessage(string userName)
    {
        lock (WeixinContextGlobal.Lock)
        {
            var messageContext = GetMessageContext(userName, true);
            return messageContext.RequestMessages.LastOrDefault();
        }
    }
    /// <summary>
    /// 获取最新一条响应数据，如果不存在，则返回 null
    /// </summary>
    /// <param name="userName">用户名（OpenId）</param>
    /// <returns></returns>
```

```
        public IResponseMessageBase GetLastResponseMessage(string userName)
        {
            lock (WeixinContextGlobal.Lock)
            {
                var messageContext = GetMessageContext(userName, true);
                return messageContext.ResponseMessages.LastOrDefault();
            }
        }
    }
```

4.3 消息处理

在完成了所有消息类型的C#数据实体定义后，XML 数据包与C#数据实体转换方法，以及用户会话上下文框架后，消息处理流程所需的基础设施已经全部搭建完毕。本节将给出一个完整的消息处理框架。

4.3.1 消息处理完整流程

微信公众平台消息的传递是基于 HTTP POST 请求实现的。微信公众号服务器的消息处理过程就是一次完整的 HTTP POST 请求响应过程，即接收微信服务器 HTTP POST 请求 （Request），调用具体的消息处理程序处理消息，将消息处理程序的结果作为 HTTP POST 请求结果（Response）回复给微信服务器。

消息的详细处理过程为：

1．在接收到微信服务器请求后，初始化一个消息处理类 MessageHandler，来负责消息处理。

2．消息处理类 MessageHandler 具有一个 Init 方法，该方法调用 XML 数据转换函数 GetRequestEntity，将含有用户消息的 XML 数据包转换为 C#数据实体 IRequestMessageBase。

3．将用户消息数据实体 IRequestMessageBase 保存到用户会话上下文的集合类 WeixinContext 中。

4．根据具体业务的需求，在执行消息处理前调用 OnExecuting 方法，对用户消息数据实体 IRequestMessageBase 进行预处理。比如，某一微信公众号有用户黑

名单机制，所有在用户黑名单中的用户，微信公众号将不回复任何消息。我们可以在 OnExecuting 方法中统一判断发送消息的用户是否在用户黑名单中，如果发现用户在黑名单中，直接退出消息处理即可。

5. 调用 MessageHandler 负责消息处理的方法 Execute 进行消息处理。Execute 方法会判断该条用户消息的消息类型，并对不同的消息类型使用不同的消息处理方法进行消息处理，返回消息处理的结果 IResponseMessageBase。

6. 将消息处理结果 IResponseMessageBase 保存到用户会话上下文的集合类 WeixinContext 中。

7. 根据具体业务需求，在执行完消息处理后调用 OnExecuted 方法，对消息处理结果 IResponseMessageBase 进行统一的特殊处理。

8. 最后调用 FillEntityWithXml 方法，将消息处理结果 IResponseMessageBase 转换为 XML 数据包格式，返回请求结果给微信服务器。

整个消息处理的完整流程如图 4-3 所示。

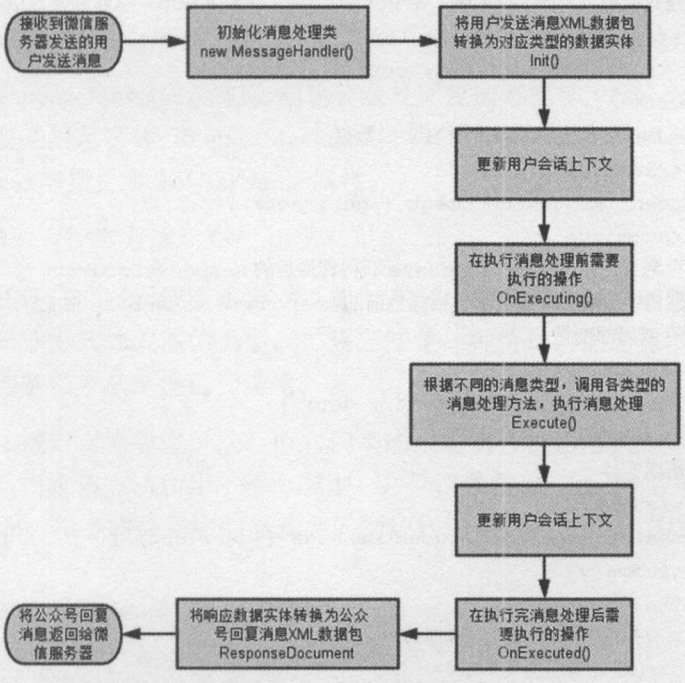


图 4-3

4.3.2 实现消息处理

根据4.3.1节对消息处理流程的分析，我们首先定义一个接口IMessageHandler，对整个消息处理流程所必须的属性与方法做出规定。在接口中，定义一个CancelExcute属性，以方便程序随时终止消息处理过程。

```
public interface IMessageHandler
{
    /// <summary>
    /// 发送者用户名（OpenId）
    /// </summary>
    string WeixinOpenId { get; }
    /// <summary>
    /// 取消执行Execute()方法。一般在OnExecuting()中用于临时阻止执行Execute()
    /// 默认为False
    /// 如果在执行OnExecuting()前设为True，则所有OnExecuting()、Execute()、
OnExecuted()代码都不会被执行
    /// 如果在执行OnExecuting()的过程中设为True，则后续Execute()及
OnExecuted()代码不会被执行
    /// 建议在设为True的时候，给ResponseMessage赋值，以返回友好信息
    /// </summary>
    bool CancelExcute { get; set; }
    /// <summary>
    /// 在构造函数中转换得到原始XML数据
    /// </summary>
    XDocument RequestDocument { get; set; }
    /// <summary>
    /// 根据ResponseMessageBase获得转换后的ResponseDocument
    /// 注意：这里每次请求都会根据当前的ResponseMessageBase生成一次，如需重用此数
据，建议使用缓存或局部变量
    /// </summary>
    XDocument ResponseDocument { get; }
    /// <summary>
    /// 请求实体
    /// </summary>
    IRequestMessageBase RequestMessage { get; set; }
    /// <summary>
    /// 响应实体
    /// 只有执行Execute()方法后才可能有值
    /// </summary>
    IResponseMessageBase ResponseMessage { get; set; }
```

```
        /// <summary>
        /// 执行微信请求的消息处理
        /// </summary>
        void Execute();
        /// <summary>
        /// 在执行消息处理前需要执行的操作
        /// </summary>
        void OnExecuting();
        /// <summary>
        /// 在执行完消息处理后需要执行的操作
        /// </summary>
        void OnExecuted();
    }
```

用户发送消息实体 RequestMessage 与公众号回复消息实体 ResponseMessage 均含有发送者 FromUserName 与接受者 ToUserName 属性，并且 RequestMessage. FromUserName 即为 ResponseMessage.ToUserName、ResponseMessage. FromUserName 即为 RequestMessage. ToUserName。

为了方便根据不同的回复类型，创建公众号回复消息实体 ResponseMessage，定义一个 IRequestMessageBase 类的扩展方法 CreateResponseMessage。

CreateResponseMessage 方法在创建公众号回复消息实体 ResponseMessage 时会根据用户发送消息实体 RequestMessage 自动填写 ResponseMessage 的发送者 FromUserName 与接受者 ToUserName 属性。

Tips1 C#语言扩展方法简介：

扩展方法使你能够向现有类型“添加”方法，而无需创建新的派生类型、重新编译或以其他方式修改原始类型。扩展方法是一种特殊的静态方法，但可以像扩展类型上的实例方法一样进行调用。

扩展方法被定义为静态方法，但它们是通过实例方法语法进行调用的。它们的第一个参数指定该方法作用于哪个类型，并且该参数以 this 修饰符为前缀。仅当你使用 using 指令将命名空间显式导入到源代码中之后，扩展方法才位于范围中。

```
    public static class EntityHelper
    {
        /// <summary>
```

```
        /// ResponseMessageBase.CreateFromRequestMessage<T>(requestMessage)
的扩展方法
        /// 初始化对应类型的公众号回复消息
        /// </summary>
        /// <typeparam name="T">需要生成的 ResponseMessage 类型</typeparam>
        /// <param name="requestMessage">IRequestMessageBase 接口下的接收信息类
型</param>
        /// <returns></returns>
        public static T CreateResponseMessage<T>(this IRequestMessageBase
requestMessage) where T : ResponseMessageBase
        {
            try
            {
                var tType = typeof(T);
                var responseName = tType.Name.Replace("ResponseMessage", ""); //
请求名称
                ResponseMsgType msgType = (ResponseMsgType)Enum.Parse(typeof
(ResponseMsgType), responseName);
                return CreateFromRequestMessage(requestMessage, msgType) as T;
            }
            catch (Exception ex)
            {
                throw new WeixinException("ResponseMessageBase.
CreateFromRequestMessage<T>过程发生异常！", ex);
            }
        }
        /// <summary>
        /// 获取响应类型实例，并初始化
        /// </summary>
        /// <param name="requestMessage">请求</param>
        /// <param name="msgType">响应类型</param>
        /// <returns></returns>
        private static ResponseMessageBase CreateFromRequestMessage
(IRequestMessageBase requestMessage, ResponseMsgType msgType)
        {
            ResponseMessageBase responseMessage = null;
            try
            {
                switch (msgType)
                {
                    case ResponseMsgType.Text:
```

```
                    responseMessage = new ResponseMessageText();
                    break;
                case ResponseMsgType.News:
                    responseMessage = new ResponseMessageNews();
                    break;
                case ResponseMsgType.Music:
                    responseMessage = new ResponseMessageMusic();
                    break;
                case ResponseMsgType.Image:
                    responseMessage = new ResponseMessageImage();
                    break;
                case ResponseMsgType.Voice:
                    responseMessage = new ResponseMessageVoice();
                    break;
                case ResponseMsgType.Video:
                    responseMessage = new ResponseMessageVideo();
                    break;
                default:
                    throw new UnknownRequestMsgTypeException(string.Format
("ResponseMsgType 没有为 {0} 提供对应处理程序。", msgType), new
ArgumentOutOfRangeException());
            }
            responseMessage.ToUserName = requestMessage.FromUserName;
            responseMessage.FromUserName = requestMessage.ToUserName;
            responseMessage.CreateTime = DateTime.Now; //使用当前最新时间
        }
        catch (Exception ex)
        {
            throw new WeixinException("CreateFromRequestMessage 过程发生异常
", ex);
        }
        return responseMessage;
    }
}
```

实现消息处理的类我们定义为 MessageHandler，该类实现接口 IMessageHandler，并包含用户会话上下文框架 IMessageContext。

```
/// <summary>
/// 微信请求的集中处理方法
/// 此方法中所有过程，都基于 Senparc.Weixin.MP 的基础功能，只为简化代码而设
/// </summary>
```

```
    public abstract class MessageHandler<TC> : IMessageHandler where TC : class,
IMessageContext, new()
    {
        /// <summary>
        /// 上下文
        /// </summary>
        protected static WeixinContext<TC> GlobalWeixinContext = new
WeixinContext<TC>();
        /// <summary>
        /// 全局消息上下文
        /// </summary>
        protected WeixinContext<TC> WeixinContext
        {
            get { return GlobalWeixinContext; }
        }
        /// <summary>
        /// 当前用户消息上下文
        /// </summary>
        public TC CurrentMessageContext
        {
            get { return WeixinContext.GetMessageContext(RequestMessage); }
        }
        /// <summary>
        /// 发送者用户名（OpenId）
        /// </summary>
        public string WeixinOpenId
        {
            get
            {
                if (RequestMessage != null)
                {
                    return RequestMessage.FromUserName;
                }
                return null;
            }
        }
        /// <summary>
        /// 取消执行 Execute()方法。一般在 OnExecuting()中用于临时阻止执行 Execute()
        /// 默认为 False
        /// 如果在执行 OnExecuting()前设为 True，则所有 OnExecuting()、Execute()、
OnExecuted()代码都不会被执行
```

```
        /// 如果在执行 OnExecuting()过程中设为 True，则后续 Execute()及 OnExecuted()
代码不会被执行
        /// 建议在设为 True 的时候，给 ResponseMessage 赋值，以返回友好信息
        /// </summary>
        public bool CancelExcute { get; set; }
        /// <summary>
        /// 在构造函数中转换得到原始 XML 数据
        /// </summary>
        public XDocument RequestDocument { get; set; }
        /// <summary>
        /// 根据 ResponseMessageBase 获得转换后的 ResponseDocument
        /// 注意：这里每次请求都会根据当前的 ResponseMessageBase 生成一次，如需重用此数
据，建议使用缓存或局部变量
        /// </summary>
        public XDocument ResponseDocument
        {
            get
            {
                if (ResponseMessage == null)
                {
                    return null;
                }
                return EntityHelper.ConvertEntityToXml(ResponseMessage as
ResponseMessageBase);
            }
        }
        /// <summary>
        /// 请求实体
        /// </summary>
        public IRequestMessageBase RequestMessage { get; set; }
        /// <summary>
        /// 响应实体
        /// 在正常情况下只有当执行 Execute()方法后才可能有值
        /// 也可以结合 Cancel，提前给 ResponseMessage 赋值
        /// </summary>
        public IResponseMessageBase ResponseMessage { get; set; }
        /// <summary>
        /// 构造函数
        /// </summary>
        /// <param name="inputStream"></param>
        /// <param name="maxRecordCount"></param>
```

```
        public MessageHandler(Stream inputStream, int maxRecordCount = 0)
        {
            WeixinContext.MaxRecordCount = maxRecordCount;
            using (XmlReader xr = XmlReader.Create(inputStream))
            {
                RequestDocument = XDocument.Load(xr);
                Init(RequestDocument);
            }
        }
        /// <summary>
        /// 构造函数
        /// </summary>
        /// <param name="requestDocument"></param>
        /// <param name="maxRecordCount"></param>
        public MessageHandler(XDocument requestDocument, int maxRecordCount = 0)
        {
            WeixinContext.MaxRecordCount = maxRecordCount;
            Init(requestDocument);
        }
        /// <summary>
        /// 执行微信请求的消息处理
        /// </summary>
        public void Execute()
        {
            //判断是否要终止消息处理过程
            if (CancelExcute)
            {
                return;
            }
            //消息预处理
            OnExecuting();
            //判断是否要终止消息处理过程
            if (CancelExcute)
            {
                return;
            }
            try
            {
                if (RequestMessage == null)
                {
                    return;
```

```
            }
            //根据具体的消息类型调用不同类型的消息处理方法来处理消息
            switch (RequestMessage.MsgType)
            {
                case RequestMsgType.Text:
                    ResponseMessage = OnTextRequest(RequestMessage as
RequestMessageText);
                    break;
                case RequestMsgType.Location:
                    ResponseMessage = OnLocationRequest(RequestMessage as
RequestMessageLocation);
                    break;
                case RequestMsgType.Image:
                    ResponseMessage = OnImageRequest(RequestMessage as
RequestMessageImage);
                    break;
                case RequestMsgType.Voice:
                    ResponseMessage = OnVoiceRequest(RequestMessage as
RequestMessageVoice);
                    break;
                case RequestMsgType.Video:
                    ResponseMessage = OnVideoRequest(RequestMessage as
RequestMessageVideo);
                    break;
                case RequestMsgType.Event:
                    ResponseMessage = OnEventRequest(RequestMessage as
RequestMessageEventBase);
                    break;
                default:
                    throw new UnknownRequestMsgTypeException("未知的 MsgType
请求类型", null);
            }
            //记录上下文
            if (WeixinContextGlobal.UseWeixinContext && ResponseMessage !=
null)
            {
                WeixinContext.InsertMessage(ResponseMessage);
            }
        }
        catch (Exception ex)
        {
```

```
                throw ex;
            }
            finally
            {
                //返回响应结果前的特殊处理
                OnExecuted();
            }
        }
        /// <summary>
        /// 在执行消息处理前需要执行的操作，视具体业务需求重写此方法
        /// </summary>
        public virtual void OnExecuting()
        {
        }
        /// <summary>
        /// 在执行完消息处理后需要执行的操作，视具体业务需求重写此方法
        /// </summary>
        public virtual void OnExecuted()
        {
        }
        /// <summary>
        /// 将用户发送消息的 XML 数据包转换为数据实体
        /// </summary>
        /// <param name="requestDocument"></param>
        private void Init(XDocument requestDocument)
        {
            RequestDocument = requestDocument;
            //将 XML 数据包转换为数据实体
            RequestMessage = RequestMessageFactory.GetRequestEntity
(RequestDocument);
            //记录上下文
            if (WeixinContextGlobal.UseWeixinContext)
            {
                WeixinContext.InsertMessage(RequestMessage);
            }
        }
        /// <summary>
        /// 根据当前的 RequestMessage 创建指定类型的 ResponseMessage
        /// </summary>
        /// <typeparam name="TR">基于 ResponseMessageBase 的响应消息类型</typeparam>
        /// <returns></returns>
```

```
        protected TR CreateResponseMessage<TR>() where TR : ResponseMessageBase
        {
            if (RequestMessage == null)
            {
                return null;
            }
            return RequestMessage.CreateResponseMessage<TR>();
        }
        /// <summary>
        /// 默认返回消息（当任何 OnXX 消息没有被重写时，都将自动返回此默认消息）
        /// </summary>
        protected virtual IResponseMessageBase DefaultResponseMessage
(IRequestMessageBase requestMessage)
        {
            var responseMessage = this.CreateResponseMessage
<ResponseMessageText>();
            responseMessage.Content = "您发送的消息类型暂未被识别。";
            return responseMessage;
        }
        #region 每种类型的消息处理方法，视具体业务需求重写每个方法
        /// <summary>
        /// 文字类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnTextRequest (RequestMessageText
requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// 位置类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnLocationRequest
(RequestMessageLocation requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// 图片类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnImageRequest (RequestMessageImage
requestMessage)
```

```
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// 语音类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnVoiceRequest(RequestMessageVoice
requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// 视频类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnVideoRequest(RequestMessageVideo
requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// 链接消息类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnLinkRequest(RequestMessageLink
requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// Event 事件类型请求
        /// </summary>
        protected virtual IResponseMessageBase OnEventRequest
(RequestMessageEventBase requestMessage)
        {
          var strongRequestMessage = RequestMessage as IRequestMessageEventBase;
           IResponseMessageBase responseMessage = null;
           switch (strongRequestMessage.Event)
           {
              case Event.LOCATION://自动发送的用户当前位置
                 responseMessage = OnEvent_LocationRequest(RequestMessage as
RequestMessageEvent_Location);
                 break;
              case Event.subscribe://订阅
```

```
                responseMessage = OnEvent_SubscribeRequest(RequestMessage as
RequestMessageEvent_Subscribe);
                break;
            case Event.unsubscribe://退订
                responseMessage = OnEvent_UnsubscribeRequest(RequestMessage
as RequestMessageEvent_Unsubscribe);
                break;
            case Event.CLICK://菜单单击
                responseMessage = OnEvent_ClickRequest(RequestMessage as
RequestMessageEvent_Click);
                break;
            case Event.scan://二维码
                responseMessage = OnEvent_ScanRequest(RequestMessage as
RequestMessageEvent_Scan);
                break;
            default:
                throw new UnknownRequestMsgTypeException("未知的 Event 下属请
求信息", null);
        }
        return responseMessage;
    }
    #endregion
    #region 每种事件消息处理方法，视具体业务需求重写每个方法
    /// <summary>
    /// Event 事件类型请求之 Locatlon
    /// </summary>
    protected virtual IResponseMessageBase OnEvent_LocationRequest
(RequestMessageEvent_Location requestMessage)
    {
        return DefaultResponseMessage(requestMessage);
    }
    /// <summary>
    /// Event 事件类型请求之 subscribe
    /// </summary>
    protected virtual IResponseMessageBase OnEvent_SubscribeRequest
(RequestMessageEvent_Subscribe requestMessage)
    {
        return DefaultResponseMessage(requestMessage);
    }
    /// <summary>
    /// Event 事件类型请求之 Unsubscribe
    /// </summary>
```

```
        protected virtual IResponseMessageBase OnEvent_UnsubscribeRequest
(RequestMessageEvent_Unsubscribe requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// Event 事件类型请求之 Click
        /// </summary>
        protected virtual IResponseMessageBase OnEvent_ClickRequest
(RequestMessageEvent_Click requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        /// <summary>
        /// Event 事件类型请求之 Scan
        /// </summary>
        protected virtual IResponseMessageBase OnEvent_ScanRequest
(RequestMessageEvent_Scan requestMessage)
        {
            return DefaultResponseMessage(requestMessage);
        }
        #endregion
    }
```

4.4 消息处理框架的完整结构

自 4.3.2 节为止，我们已经实现了一个微信公众号消息处理框架的所有代码。为了便于在实际开发中使用该消息处理框架，在 Microsoft Visual Studio 2012 开发环境中建立一个类库项目 WeiXinMessageSDK，将整个消息处理框架代码整理到该项目中。

WeiXinMessageSDK 项目的结构如图 4-4 所示。

Context 文件夹用于存放用户会话上下文框架的代码（4.2 节）。

Entities 文件夹中包含所有类型的用户发送消息和公众号回复消息的数据实体的定义，其中的 Request 文件夹用于存放用户发送消息数据实体，Response 文件夹用于存放公众号回复消息数据实体（4.1.3 节、4.1.4 节、4.1.5 节）。

Exceptions 文件夹包含消息处理框架的错误类型定义（4.1.6 节）。

Helpers 文件夹包含了一些进行数据类型转换的方法。消息的 XML 数据包与 C#数据实体的转换方法 FillEntityWithXml 和 ConvertEntityToXml 被整合进 EntityHelper 类中（4.1.6 节、4.3.2 节）。

MessageHandlers 文件夹中保存了具体实现整个消息处理流程的方法(4.3.2 节)。

CheckSignature.cs 为微信公众平台的接口校验方法（第 3 章）。

Enums.cs 中包含所有的用户发送消息与公众号回复消息的类型枚举定义（4.1.3 节、4.1.4 节、4.1.5 节）。

RequestMessageFactory.cs 中包含将任意类型用户发送消息 XML 数据包转换为对应数据实体的方法（4.1.6 节）。

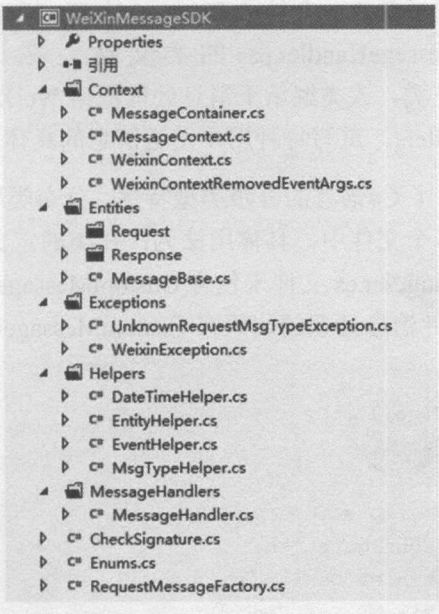


图 4-4

4.5　消息处理框架使用示例

WeiXinMessageSDK 提供了一个与具体业务无关的微信公众平台消息处理基

础框架，其中并没有包含如何具体处理每种类型消息的业务逻辑。在实际开发中，我们需要根据具体业务需求，继承 WeiXinMessageSDK 的消息处理类 MessageHandler，重写每种消息处理的具体方法。

4.5.1 消息处理框架示例程序

下面我们以一个实际例子来讲解如何使用消息处理框架 WeiXinMessageSDK 快速实现微信公众平台的消息处理。

1．首先打开 Microsoft Visual Studio 2012 开发环境，建立一个名为 Sample_2 的 ASP.NET 空 Web 应用程序。

2．在 Sample_2 项目中引用 WeiXinMessageSDK。

3．在 Sample_2 项目新建一个目录 CustomMessageHandler，在该目录中新建一个名为 CustomMessageHandler.cs 的类文件。在该文件中定义一个 CustomMessageHandler 类，该类继承于消息处理框架 WeiXinMessageSDK 中的消息处理类 MessageHandlers，重写每种用户发送消息的具体处理方法。

在这里，我们使用了 C#语言的分步类型特性，分布类型定义允许将类、结构或接口的定义拆分到多个文件中，具体用法为：在类的定义中加上“Partical”关键字。CustomMessageHandler.cs 文件未包含 CustomMessageHandler 类关于事件消息处理的具体代码，事件消息处理的代码在 CustomMessageHandler_Events.cs 中。

```
using System;
using System.Configuration;
using System.IO;
using System.Text;
using System.Collections.Generic;
using System.Web.Configuration;
using Senparc.Weixin.MP.Context;
using Senparc.Weixin.MP.Entities;
using Senparc.Weixin.MP.MessageHandlers;
using Senparc.Weixin.MP.Helpers;
namespace Sample_2
{
    /// <summary>
    /// 自定义MessageHandler
    /// 把MessageHandler作为基类，重写对应请求的处理方法
```

```
        /// </summary>
        public partial class CustomMessageHandler : MessageHandler <MessageContext>
        {
            /// <summary>
            /// 构造函数
            /// </summary>
            /// <param name="inputStream"></param>
            /// <param name="maxRecordCount"></param>
            public CustomMessageHandler(Stream inputStream, int maxRecordCount
= 0)
                : base(inputStream, maxRecordCount)
            {
                //这里设置仅用于测试，实际开发可以在外部更全局的地方设置
                // 比 如  MessageHandler<MessageContext>.GlobalWeixinContext.
ExpireMinutes = 3
                WeixinContext.ExpireMinutes = 3;
            }
            public override void OnExecuting()
            {
                //测试 MessageContext.StorageData
                if (CurrentMessageContext.StorageData == null)
                {
                    CurrentMessageContext.StorageData = 0;
                }
                base.OnExecuting();
            }
            public override void OnExecuted()
            {
                base.OnExecuted();
                CurrentMessageContext.StorageData                              =
((int)CurrentMessageContext.StorageData) + 1;
            }
            /// <summary>
            /// 处理文字请求
            /// </summary>
            /// <returns></returns>
            protected override IResponseMessageBase OnTextRequest
(RequestMessageText requestMessage)
            {
                //注意：下面泛型 ResponseMessageText 即返回给客户端的类型，可以根据自己
的需要填写 ResponseMessageNews 等不同类型
```

```
            var responseMessage = CreateResponseMessage<ResponseMessageText>();

             var result = new StringBuilder();
             result.AppendFormat("您刚才发送了文字信息：{0}\r\n\r\n",
requestMessage.Content);

             if (CurrentMessageContext.RequestMessages.Count > 1)
             {
                result.AppendFormat("您刚才还发送了如下消息（{0}/{1}）：\r\n",
CurrentMessageContext.RequestMessages.Count, CurrentMessageContext.StorageData);
                for (int i = CurrentMessageContext.RequestMessages.Count - 2;
i >= 0; i--)
                {
                 var historyMessage = CurrentMessageContext.RequestMessages[i];
                   result.AppendFormat("{0} 【{1}】{2}\r\n",
                            historyMessage.CreateTime.ToShortTimeString(),
historyMessage.MsgType.ToString(), (historyMessage is RequestMessageText)?
(historyMessage as RequestMessageText). Content: "[非文字类型]"
                        );
                }
                result.AppendLine("\r\n");
             }
             result.AppendFormat("如果您在{0}分钟内连续发送消息，记录将被自动保留
（当前设置：最多记录{1}条）。过期后记录将会自动清除。\r\n", WeixinContext.ExpireMinutes,
WeixinContext.MaxRecordCount);
             result.AppendLine("\r\n");
             result.AppendLine("您还可以发送【位置】【图片】【语音】【视频】等类型的信
息（注意是这几种类型，不是这几个文字），查看不同格式的回复。");
             responseMessage.Content = result.ToString();
             return responseMessage;
          }
          /// <summary>
          /// 处理位置请求
          /// </summary>
          /// <param name="requestMessage"></param>
          /// <returns></returns>
          protected override IResponseMessageBase OnLocationRequest
(RequestMessageLocation requestMessage)
          {
            var responseMessage = CreateResponseMessage<ResponseMessageText>();
```

```
            responseMessage.Content = string.Format("您刚才发送了地理位置信息。
Location_X:{0},Location_Y:{1},Scale:{2},标签:{3}",requestMessage.Location_X,
requestMessage.Location_Y,requestMessage.Scale, requestMessage.Label);
            return responseMessage;
        }
        /// <summary>
        /// 处理图片请求
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnImageRequest
(RequestMessageImage requestMessage)
        {
          var responseMessage = CreateResponseMessage<ResponseMessageNews>();
            responseMessage.Articles.Add(new Article()
            {
                Title = "您刚才发送了图片信息",
                Description = "您发送的图片将会显示在边上",
                PicUrl = requestMessage.PicUrl,
                Url = "http://www.qxuninfo.com"
            });
            responseMessage.Articles.Add(new Article()
            {
                Title = "第二条",
                Description = "第二条带链接的内容",
                PicUrl = requestMessage.PicUrl,
                Url = "http://www.qxuninfo.com"
            });
            return responseMessage;
        }
        /// <summary>
        /// 处理语音请求
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnVoiceRequest
(RequestMessageVoice requestMessage)
        {
          var responseMessage = CreateResponseMessage <ResponseMessageMusic>();
            responseMessage.Music.MusicUrl   =   "http://www.qxuninfo.com/
music.mp3";
```

```
            responseMessage.Music.Title = "这里是一条音乐消息";
            responseMessage.Music.Description = "时间都去哪儿了";
            return responseMessage;
        }
        /// <summary>
        /// 处理视频请求
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnVideoRequest
(RequestMessageVideo requestMessage)
        {
          var responseMessage = CreateResponseMessage <ResponseMessageText>();
            responseMessage.Content = "您发送了一条视频信息，ID: " +
requestMessage.MediaId;
            return responseMessage;
        }
        /// <summary>
        /// 处理链接消息请求
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnLinkRequest
(RequestMessageLink requestMessage)
        {
          var responseMessage = CreateResponseMessage<ResponseMessageText>();
            responseMessage.Content = string.Format(@"您发送了一条链接信息:
    Title: {0}
    Description:{1}
    Url:{2}", requestMessage.Title, requestMessage.Description, requestMessage.
Url);
            return responseMessage;
        }
        /// <summary>
        /// 处理事件请求（这个方法一般不用重写，这里仅作为示例出现。除非需要在判断具体
Event 类型以外对 Event 信息进行统一操作
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnEventRequest
(RequestMessageEventBase requestMessage)
```

```
        {
            var eventResponseMessage = base.OnEventRequest(requestMessage);
//对于 Event 下属分类的重写方法，见：CustomerMessageHandler_Events.cs
            return eventResponseMessage;
        }
        protected override IResponseMessageBase DefaultResponseMessage
(IRequestMessageBase requestMessage)
        {
            //所有没有被处理的消息会默认返回这里的结果
          var responseMessage = this.CreateResponseMessage<ResponseMessageText>
();
            responseMessage.Content = "这条消息来自 DefaultResponseMessage。";
            return responseMessage;
        }
    }
}
```

1．在目录 CustomMessageHandler 中新建一个名为 CustomMessageHandler_Events.cs 的类文件。该文件包含了 CustomMessageHandler 类对每种事件消息处理的具体代码。

```
using System;
using System.Diagnostics;
using System.Linq;
using System.Web;
using Senparc.Weixin.MP.Context;
using Senparc.Weixin.MP.Entities;
using Senparc.Weixin.MP.Helpers;
using Senparc.Weixin.MP.MessageHandlers;
namespace Sample_2
{
    /// <summary>
    /// 自定义 MessageHandler
    /// </summary>
    public partial class CustomMessageHandler : MessageHandler <MessageContext>
    {
        /// <summary>
        /// 微信客户端（通过微信服务器）自动发送过来的位置信息事件
        /// </summary>
        /// <param name="requestMessage"></param>
        /// <returns></returns>
        protected override IResponseMessageBase OnEvent_LocationRequest
```

```
(RequestMessageEvent_Location requestMessage)
        {
          var responseMessage = CreateResponseMessage<ResponseMessageText>();
           responseMessage.Content = "这里写什么都无所谓，比如：上帝爱你！";
           return responseMessage;//这里也可以返回 null（需要注意写日志时候 null
的问题）
        }
        /// <summary>
        /// 订阅（关注）事件
        /// </summary>
        /// <returns></returns>
        protected override IResponseMessageBase OnEvent_SubscribeRequest
(RequestMessageEvent_Subscribe requestMessage)
        {
          var responseMessage = CreateResponseMessage<ResponseMessageText>();
           responseMessage.Content = @"您可以发送【文字】【位置】【图片】【语音】等
不同类型的信息，查看不同格式的回复。您也可以直接单击菜单查看各种类型的回复。";
           return responseMessage;
        }
        /// <summary>
        /// 退订
        /// 实际上用户无法收到非订阅账号的消息，所以这里可以随便写
        /// Unsubscribe 事件的意义在于及时删除网站应用中已经记录的 OpenId 绑定，消除
冗余数据。并且关注用户流失的情况
        /// </summary>
        /// <returns></returns>
        protected override IResponseMessageBase OnEvent_UnsubscribeRequest
(RequestMessageEvent_Unsubscribe requestMessage)
        {
           var responseMessage = base.CreateResponseMessage
<ResponseMessageText>();
           responseMessage.Content = "有空再来";
           return responseMessage;
        }
        /// <summary>
        /// 自定义菜单单击事件
        /// </summary>
        /// <returns></returns>
        protected override IResponseMessageBase OnEvent_ClickRequest
(RequestMessageEvent_Click requestMessage)
        {
```

```
            IResponseMessageBase reponseMessage = null;
            //菜单单击，需要跟创建菜单时的 Key 匹配
            switch (requestMessage.EventKey)
            {
                case "OneClick":
                    {
                        var strongResponseMessage = CreateResponseMessage
<ResponseMessageText>();
                        strongResponseMessage.Content = "您单击了底部按钮。";
                        reponseMessage = strongResponseMessage;
                    }
                    break;
                case "SubClickRoot_Text":
                    {
                        var strongResponseMessage = CreateResponseMessage
<ResponseMessageText>();
                        strongResponseMessage.Content = "您单击了子菜单按钮。";
                        reponseMessage = strongResponseMessage;
                    }
                    break;
            }
            return reponseMessage;
        }
        /// <summary>
        /// 扫描带参数二维码事件
        /// 实际上用户无法收到非订阅账号的消息，所以这里可以随便写
        /// Scan 事件的意义在于获取扫描二维码中包含的参数，便于识别和统计用户扫描了哪
个二维码
        /// </summary>
        /// <returns></returns>
        protected override IResponseMessageBase OnEvent_ScanRequest
(RequestMessageEvent_Scan requestMessage)
        {
            var responseMessage = base.CreateResponseMessage
<ResponseMessageText>();
            responseMessage.Content = "感谢扫码";
            return responseMessage;
        }
    }
}
```

2．最后，在 Sample_2 项目下创建一个名为 Index.aspx 的 Web 窗体，该 Web 窗体负责接收微信服务器的消息请求，调用 CustomMessageHandler 类处理消息，并将消息处理结果返回给微信服务器。

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Senparc.Weixin.MP;
namespace Sample_2
{
    public partial class Index: System.Web.UI.Page
    {
        private readonly string Token = "weixin";//与微信公众账号后台的 Token
设置保持一致，区分大小写
        protected void Page_Load(object sender, EventArgs e)
        {
            string signature = Request["signature"];
            string timestamp = Request["timestamp"];
            string nonce = Request["nonce"];
            string echostr = Request["echostr"];
            if(Request.HttpMethod == "GET")
            {
                //get method 仅在微信后台填写 URL 验证时触发
                if(CheckSignature.Check(signature, timestamp, nonce, Token))
                {
                    WriteContent(echostr); //返回随机字符串则表示验证通过
                }
                else
                {
                    WriteContent("failed:" + signature + "," + CheckSignature.
GetSignature(timestamp, nonce, Token)+"。"+
                                 "如果你在浏览器中看到这句话，说明此地址可以被作为微信公
众账号后台的 URL，请注意保持 Token 一致。");
                }
                Response.End();
            }
            else
```

```
            {
                //post method 当有用户需要公众账号发送消息时触发
                if(!CheckSignature.Check(signature, timestamp, nonce, Token))
                {
                    WriteContent("参数错误！");
                    return;
                }
                //设置每个人上下文消息存储的最大数量，防止内存占用过多，如果该参数小于等于0，则不限制
                var maxRecordCount = 10;
                //自定义MessageHandler，对微信请求的详细判断操作都在这里面
                var messageHandler = new CustomMessageHandler(Request.InputStream, maxRecordCount);
                try
                {
                    //测试时可开启此记录，帮助跟踪数据，使用前请确保App_Data文件夹存在，且有读写权限
                    messageHandler.RequestDocument.Save(
                        Server.MapPath("~/App_Data/" + DateTime.Now.Ticks + "_Request_" +
                                        messageHandler.RequestMessage.FromUserName + ".txt"));
                    //执行微信处理过程
                    messageHandler.Execute();
                    //测试时可开启，帮助跟踪数据
                    messageHandler.ResponseDocument.Save(
                        Server.MapPath("~/App_Data/" + DateTime.Now.Ticks + "_Response_" +
                                        messageHandler.ResponseMessage.ToUserName + ".txt"));
                    WriteContent(messageHandler.ResponseDocument.ToString());
                    return;
                }
                catch (Exception ex)
                {
                    //将程序运行中发生的错误记录到App_Data文件夹中
                    using (TextWriter tw = new StreamWriter(Server.MapPath("~/App_Data/Error_" + DateTime.Now.Ticks + ".txt")))
                    {
                        tw.WriteLine(ex.Message);
                        tw.WriteLine(ex.InnerException.Message);
```

```
                    if (messageHandler.ResponseDocument != null)
                    {
                        tw.WriteLine(messageHandler.ResponseDocument.
ToString());
                    }
                    tw.Flush();
                    tw.Close();
                }
                WriteContent("");
            }
            finally
            {
                Response.End();
            }
        }
    }
    private void WriteContent(string str)
    {
        Response.Output.Write(str);
    }
  }
}
```

Sample_2 项目的完整结构如图 4-5 所示。



图 4-5

使用 Microsoft Visual Studio 2012 打开 Sample_2 项目，按“F5”启动项目调试，会自动打开浏览器。如果看到如图 4-10 所示的界面，则表示项目已能正常运行。

4.5.2　在 AppHarbor 部署示例程序

在 Sample_2 项目本地调试通过后，需要将项目上传到 AppHarbor 中进行部署。让微信服务器能访问到我们的程序，进行实际测试。

首先删除“D:/weixintest”文件夹下的所有文件，然后打开“Git Bash”命令行工具，依次输入以下命令，获取 AppHarbor 中当前版本文件列表，如图 4-6 所示。

```
cd d:/weixintest/
git remote add appharbor https://weixintest001@appharbor.com/weixintest-1.git
git pull appharbor master
```

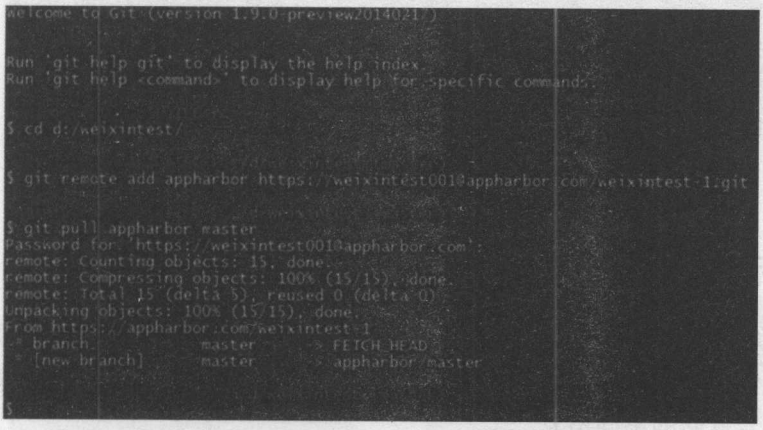


图 4-6

然后将需要上传的项目文件复制到“D:/weixintest”文件夹中。Sample_2 项目需要上传的项目文件有：Bin 文件夹、App_Data 文件夹、Index.aspx 文件、web.config 文件。在“Git Bash”命令行工具中依次输入以下命令，将 Sample_2 项目上传到 AppHarbor 中，如图 4-7 所示。

```
git add -A
git commit -m "Sample_2"
git push appharbor master
```

在浏览器中打开网址“http://weixintest-1.apphb.com/index.aspx”，我们将看到图 3-10 中的提示信息，这表明我们的 Sample_2 项目已能被微信服务器正常访问，也能正确处理 GET 请求。

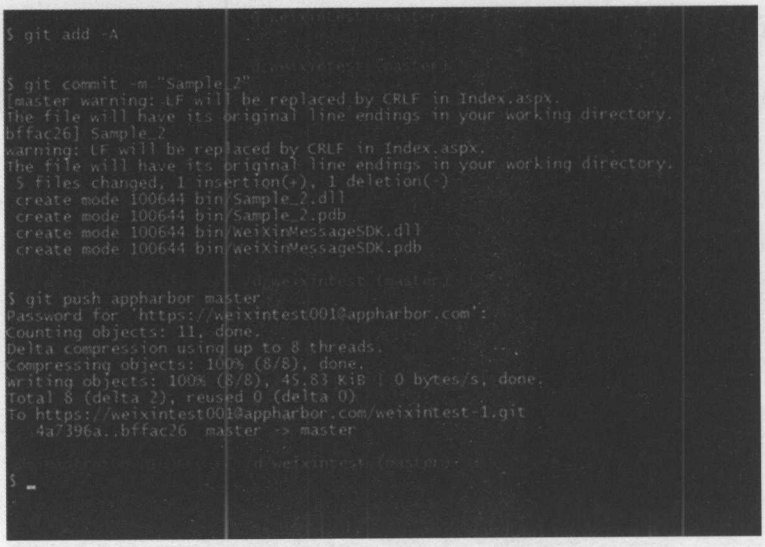


图 4-7

4.5.3 示例程序运行结果

将 Sample_2 项目部署到 AppHarbor 上，并接入微信公众平台后，使用微信客户端关注微信公众号。向该微信公众号发送不同类型的消息，应该能正确接收到微信公众号返回的不同消息。

以下为一些微信客户端发送消息及微信公众号回复消息截图。

图 4-8 为关注微信公众号时，微信公众号针对关注事件（Event.subscribe）的文本消息（ResponseMsgType.Text）回复截图。

图 4-9 为微信客户端发送文字消息时，微信公众号针对文字消息（RequestMsgType.Text）的文本消息（ResponseMsgType.Text）回复截图。

图 4-10 为微信客户端发送地理位置消息时，微信公众号针对地理位置消息（RequestMsgType. Location）的文本消息（ResponseMsgType.Text）回复截图。

图 4-11 为微信客户端发送语音消息时，微信公众号针对语音消息（RequestMsgType. Voice）的音乐消息（ResponseMsgType.Music）回复截图。

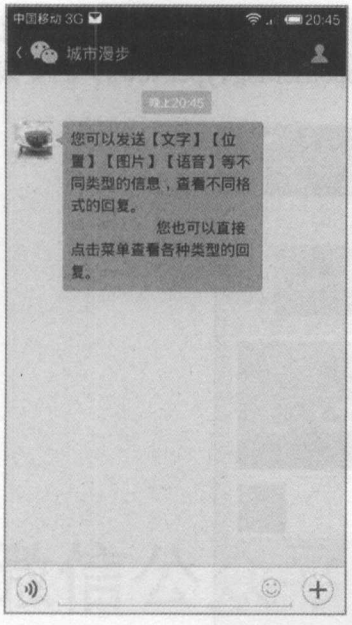


图 4-8

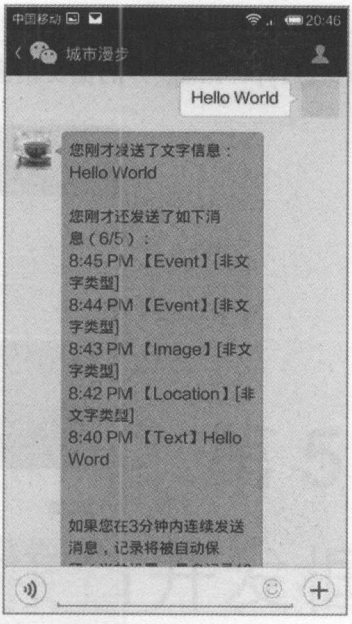


图 4-9

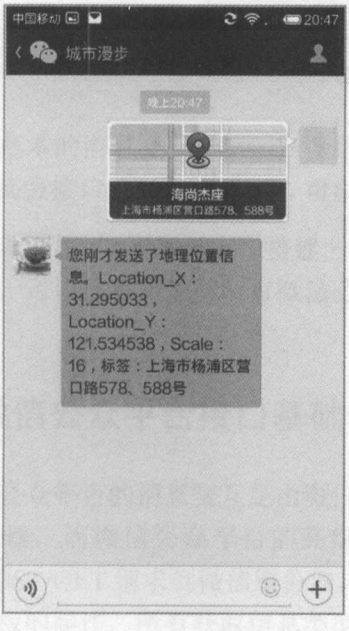


图 4-10

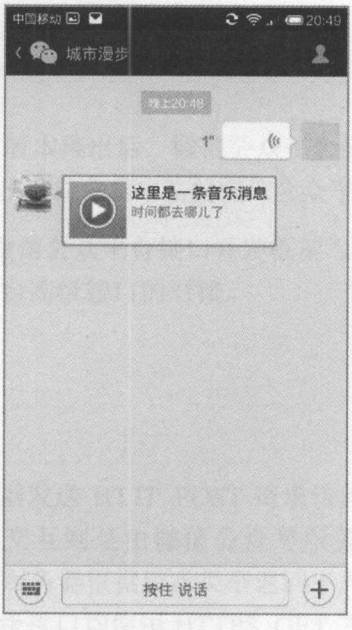


图 4-11

图 4-12 为微信客户端发送地理位置消息时，微信公众号针对地理位置消息（RequestMsgType. Image）的图文消息（ResponseMsgType.News）回复截图。

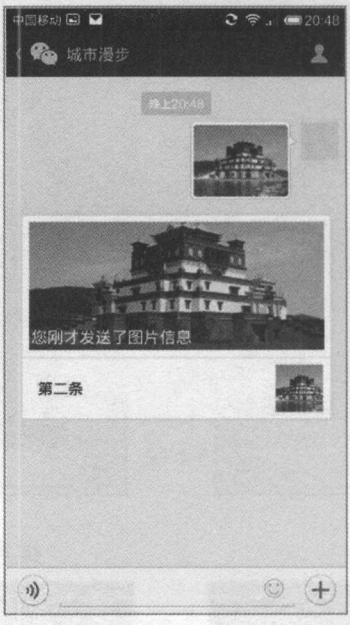


图 4-12

第5章
微信公众平台接口开发框架

除了基本的消息交互接口，在微信 5.0 版本推出后，微信公众平台还开放了一系列的高级接口。利用这些接口，可以开发出功能非常丰富的微信公众平台应用。

本章将利用 C#开发语言，搭建一个“微信公众平台接口开发框架”，基于该开发框架，可以快速实现与所有微信公众平台高级接口的对接。

5.1 微信公众平台接口基础

微信公众平台的消息交互是由微信服务器发送 HTTP POST 请求访问微信公众号服务器，而微信公众平台的高级接口交互则是由微信公众号服务器发送 HTTPS POST/GET 请求到微信服务器，微信服务器根据请求类型返回不同的信息或执行相应的操作，所有获取信息类型的高级接口均使用 HTTPS GET 方式，而执行操作的高级接口均使用 HTTPS POST。

5.1.1 高级接口交互流程

为了确保接口的通信安全性，与消息接口的 Token 验证机制类似，所有高级接口的访问，微信都规定了一个 acccess_token 参数来进行访问权限验证。

access_token 是微信公众号的全局唯一票据，微信公众号调用各高级接口时都需要使用 access_token。在正常情况下 access_token 有效期为 7200 秒，重复获取将导致上次获取的 access_token 失效。

微信公众号可以使用 AppId 和 AppSecret 来获取 access_token。AppId 和 AppSecret 可在“开发者中心”获得。当公众号成为开发者后，登录微信公众平台，进入“开发者中心”，在“开发者 ID”栏目中显示的就是当前微信公众号的 AppId 和 AppSecret，如图 5-1 所示。

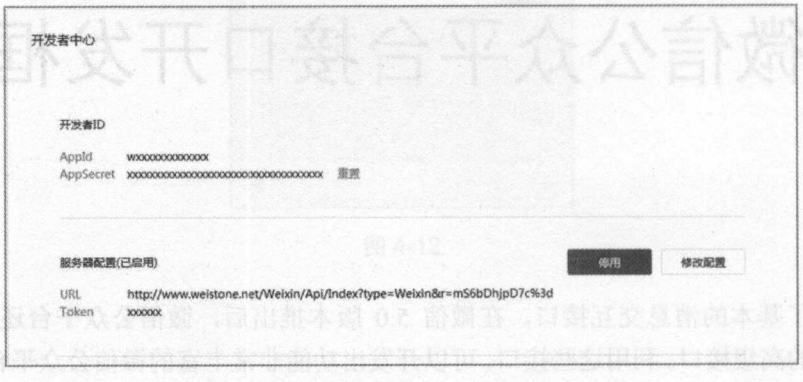


图 5-1

图 5-2 展示了一次完整的高级接口交互流程。

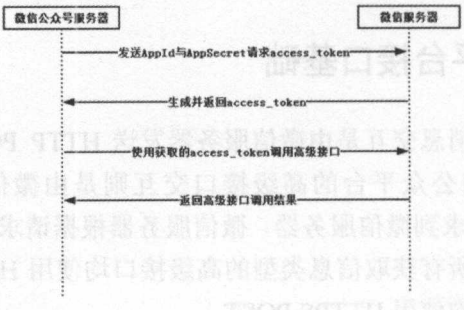


图 5-2

5.1.2　实现 HTTPS 请求

为了能访问高级接口，我们首先需要定义两个方法来实现 HTTPS GET 请求和 HTTPS POST 请求。

在 C#中，有两种方式来实现 HTTP/HTTPS 请求：

1．使用 System.Net 命名空间中的 WebClient 类，该类包含了 HTTP/HTTPS 请求的常用方法，只需要简单调用方法即可实现 HTTP/HTTPS 请求。

2．使用 System.Net 命名空间中的 HttpWebRequest 与 HttpWebResponse 类，这两个类只实现了基本的 HTTP/HTTPS 协议操作。使用这两个类实现 HTTP/HTTPS 请求，需要了解 HTTP 协议的基本知识。相比 WebClient，使用 HttpWebRequest 与 HttpWebResponse 能更灵活地操作 HTTP/HTTPS 请求，但程序实现上比较复杂。

HTTPS GET 请求我们采用 WebClient 实现，而 HTTPS POST 请求采用 HttpWebRequest/ HttpWebResponse 的方式实现，使读者对两种实现方式都能有所了解。

新建一个 HTTP 访问工具类 RequestUtility，HttpGet 方法实现 HTTPS GET 请求，HttpPost 方法实现 HTTPS POST 请求。

在访问高级接口时，默认的 HTTPS 请求编码方式（Encoding）为 UTF8。

```
using System;
using System.IO;
using System.Net;
using System.Text;

namespace Senparc.Weixin.MP.HttpUtility
{
    /// <summary>
    /// HTTPS GET/POST 方法
    /// </summary>
    public static class RequestUtility
    {
        /// <summary>
        /// WebClient 实现 HTTPS GET 方法
        /// </summary>
        /// <param name="url"></param>
        /// <returns></returns>
```

```
        public static string HttpGet(string url, Encoding encoding = null)
        {
            WebClient wc = new WebClient();
            wc.Encoding = encoding ?? Encoding.UTF8;
            //GET 请求
            return wc.DownloadString(url);
        }
        /// <summary>
        /// HttpWebRequest/HttpWebResponse 实现 HTTPS POST 方法
        /// </summary>
        /// <param name="url"></param>
        /// <param name="postStream">POST 提交的数据</param>
        /// <returns></returns>
        public static string HttpPost(string url, Stream postStream,
Encoding encoding = null)
        {
            //创建 HTTP 请求
            HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
            //定义 HTTP 请求类型
            request.Method = "POST";
            #region 定义 HTTP 头信息
            request.ContentType = "application/x-www-form-urlencoded";
            request.ContentLength = postStream != null ? postStream.Length : 0;
            request.Accept = "text/html,application/xhtml+xml,application/xml;
q=0.9,image/webp,*/*;q=0.8";
            request.KeepAlive = true;
            request.UserAgent = "Mozilla/5.0 (Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.57 Safari/537.36";
            #endregion
            #region 将要 POST 提交的数据输入二进制流
            if (postStream != null)
            {
                postStream.Position = 0;

                //直接写入流
                Stream requestStream = request.GetRequestStream();

                byte[] buffer = new byte[1024];
                int bytesRead = 0;
                while ((bytesRead = postStream.Read(buffer, 0, buffer.Length)) !=
0)
```

```
                {
                    requestStream.Write(buffer, 0, bytesRead);
                }

                postStream.Close();//关闭文件访问
            }
            #endregion
            //进行 HTTP 请求，并获取 HTTP 请求返回的结果
            HttpWebResponse response = (HttpWebResponse)request. GetResponse();
            //将返回结果的二进制流转换为字符串结果
            using (Stream responseStream = response.GetResponseStream())
            {
                using (StreamReader myStreamReader = new StreamReader
(responseStream, encoding ?? Encoding.GetEncoding("utf-8")))
                {
                    string retString = myStreamReader.ReadToEnd();
                    return retString;
                }
            }
        }
    }
}
```

5.1.3　封装接口访问方法

在建立好通用的 HTTPS GET/POST 请求方法后，还需要根据微信公众平台高级接口的特点，实现一套适用于高级接口应用场景的接口访问方法，这套方法需要统一高级接口访问的格式，能对高级接口的请求发送和返回结果进行统一处理。规范和统一高级接口访问方法，一方面可以减少在编写各个接口具体访问方法时可能出现的错误数量，另一方面通过面向对象的继承思想，大大减少编写代码的行数，增加程序可读性，便于以后维护及扩展。

通过阅读《微信公众平台开发者文档》中所有高级接口的使用说明，提炼出高级接口的共同特点：

1. 接口访问出错，都将会返回同样格式的错误信息，每种错误都有对应的错误代码(errcode)与错误消息提示(errmsg)，如：“{"errcode":40004,"errmsg":"invalid media type"}”。

2．所有获取信息类型的高级接口使用 HTTPS GET 方式，而执行操作的高级接口均使用 HTTPS POST。

3．执行操作的高级接口，如果操作成功，那么返回结果为：“{"errcode":0,"errmsg":"ok"}”。

4．接口参数都含有 access_token。

首先，根据特点 1、3 定义出接口返回结果基类 WxJsonResult，建立一个枚举 ReturnCode 来统一表示所有的错误信息，以及建立一个高级接口访问出错的自定义错误类型 ErrorJsonResultException。所有高级接口访问出错，统一抛出 ErrorJsonResultException 类型的错误，便于通过调试和读取错误日志快速发现与解决错误。

```
/// <summary>
/// 微信高级接口返回的结果
/// 正确时返回的 JSON 数据包如下：
/// {"errcode":0,"errmsg":"ok"}
/// </summary>
public class WxJsonResult
{
    /// <summary>
    /// 错误类型
    /// </summary>
    public  ReturnCode errcode { get; set; }
    /// <summary>
    /// 错误提示信息
    /// </summary>
    public string errmsg { get; set; }
}
/// <summary>
/// 微信接口返回错误类型
/// 所有的官方错误类型定义参见
/// http://mp.weixin.qq.com/wiki/index.php?title=全局返回码说明
/// </summary>
public enum ReturnCode
{
    系统繁忙 = -1,
    请求成功 = 0,
    验证失败 = 40001,
    .
```

```
        .
        .
        用户未授权该 api = 50001
    }
    /// <summary>
    /// 微信高级接口访问类型错误
    /// </summary>
    public class ErrorJsonResultException : WeixinException
    {
        public WxJsonResult JsonResult { get; set; }
        public ErrorJsonResultException(string message, Exception inner,
WxJsonResult jsonResult)
            : base(message, inner)
        {
            JsonResult = jsonResult;
        }
    }
```

然后，定义一个 ApiHelper 类，将访问高级接口的 HTTPS POST/GET 方法统一在该类中实现。获取信息类型的高级接口调用 public static T Get<T>(string accessToken, string urlFormat, params string[] querys)方法，执行操作的高级接口调用 public static T Post<T>(string accessToken, string urlFormat, object data, params string[] querys)方法。

```
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Text;
    using System.Web.Script.Serialization;
    using Senparc.Weixin.MP.Entities;
    using Senparc.Weixin.MP.HttpUtility;

    namespace Senparc.Weixin.MP.Helpers
    {
       public static class ApiHelper
       {
          /// <summary>
          /// 从微信公众平台 API 获取信息的公共方法
          /// </summary>
          /// <param name="urlFormat">API 接口地址格式</param>
```

```
        /// <param name="accessToken">微信公众号访问授权 accessToken</param>
        /// <param name="querys">除了 accessToken 还需要传递的其他参数</param>
        /// <returns></returns>
        public static WxJsonResult Get(string accessToken, string urlFormat,
params string[] querys)
        {
            return Get<WxJsonResult>(urlFormat, accessToken, querys);
        }
        /// <summary>
        /// 从微信公众平台 API 获取信息的公共方法
        /// </summary>
        /// <param name="urlFormat">API 接口地址格式</param>
        /// <param name="accessToken">微信公众号访问授权 accessToken</param>
        /// <param name="querys">除了 accessToken 还需要传递的其他参数</param>
        /// <returns></returns>
        public static T Get<T>(string accessToken, string urlFormat, params
string[] querys)
        {
            var url = GetApiUrl(urlFormat, accessToken, querys);
            string result = RequestUtility.HttpGet(url, null);
            return GetResult<T>(result);
        }
        /// <summary>
        /// 向微信公众平台 API 发送信息的公共方法
        /// </summary>
        /// <param name="urlFormat">API 接口地址格式</param>
        /// <param name="accessToken">微信公众号访问授权 accessToken</param>
        /// <param name="data">POST 提交的数据</param>
        /// <param name="querys">除了 accessToken 还需要传递的其他参数</param>
        /// <returns></returns>
        public static WxJsonResult Post(string accessToken, string urlFormat,
object data, params string[] querys)
        {
            return   Post<WxJsonResult>(urlFormat,   accessToken,   data,
querys);
        }
        /// <summary>
        /// 向微信公众平台 API 发送信息的公共方法
        /// </summary>
        /// <param name="urlFormat">API 接口地址格式</param>
        /// <param name="accessToken">微信公众号访问授权 accessToken</param>
```

```
        /// <param name="data">POST 提交的数据</param>
        /// <param name="querys">除了 accessToken 还需要传递的其他参数</param>
        /// <returns></returns>
        public static T Post<T>(string accessToken, string urlFormat, object
data, params string[] querys)
        {
            var url = GetApiUrl(urlFormat, accessToken, querys);
             JavaScriptSerializer js = new JavaScriptSerializer();
             var jsonString = js.Serialize(data);
            using (MemoryStream ms = new MemoryStream())
            {
                var bytes = Encoding.UTF8.GetBytes(jsonString);
                ms.Write(bytes, 0, bytes.Length);
                ms.Seek(0, SeekOrigin.Begin);
                string result = RequestUtility.HttpPost(url, ms, null);
                return GetResult<T>(result);
            }
        }
        /// <summary>
        /// 获取 API 返回结果
        /// </summary>
        /// <typeparam name="T"></typeparam>
        /// <param name="returnText"></param>
        /// <returns></returns>
        public static T GetResult<T>(string returnText)
        {
            JavaScriptSerializer js = new JavaScriptSerializer();
            if (returnText.Contains("errcode"))
            {
                //可能发生错误
                WxJsonResult errorResult = js.Deserialize<WxJsonResult>
(returnText);
                if (errorResult.errcode != ReturnCode.请求成功)
                {
                    //发生错误
                    throw new ErrorJsonResultException(
                        string.Format("微信 Post 请求发生错误！错误代码：{0}，说明：
{1}",(int)errorResult.errcode,errorResult.errmsg),null, errorResult);
                }
            }
            T result = js.Deserialize<T>(returnText);
```

```
            return result;
        }
        /// <summary>
        /// 生成微信公众平台 API 访问地址
        /// </summary>
        /// <param name="urlFormat">API 接口地址格式</param>
        /// <param name="accessToken">微信公众号访问授权 accessToken</param>
        /// <param name="querys">除了 accessToken 还需要传递的其他参数</param>
        /// <returns></returns>
        private static string GetApiUrl(string urlFormat, string accessToken,
string[] querys)
        {
            string[] args = new string[] { accessToken };
            if (querys.Length > 0)
                args = args.Concat(querys).ToArray();
            var url = string.Format(urlFormat, args);
            return url;
        }
    }
}
```

5.2 获取接口访问凭证

所有高级接口的访问，需要先获取接口访问凭证 access_token。获取 access_token 的 API 地址为：

```
https://api.weixin.qq.com/cgi-bin/token?grant_type=client_credential&a
ppid=APPID&secret=APPSECRET
```

调用 access_tokenAPI 的程序实现：

```
/// <summary>
/// access_token 请求后的 JSON 返回格式
/// </summary>
public class AccessTokenResult
{
    /// <summary>
    /// 获取到的凭证
    /// </summary>
    public string access_token { get; set; }
```

```
        /// <summary>
        /// 凭证有效时间，单位：秒
        /// </summary>
        public int expires_in { get; set; }
    }
    /// <summary>
    /// 获取微信 access_token
    /// </summary>
    public class Token
    {
        /// <summary>
        /// 获取凭证接口
        /// </summary>
        /// <param name="grant_type">获取 access_token 填写 client_credentia
</param>
        /// <param name="appid">第三方用户唯一凭证</param>
        /// <param name="secret">第三方用户唯一凭证密钥，即 appsecret</param>
        /// <returns></returns>
        public static AccessTokenResult GetToken(string appid, string secret,
string grant_type = "client_credential")
        {
           //微信公众平台获取 access_token 接口地址
            string url = string.Format("https://api.weixin.qq.com/cgi-bin/
token?grant_type={0}&appid={1}&secret={2}",
                                       grant_type, appid, secret);
            string result = RequestUtility.HttpGet(url, null);
            return ApiHelper.GetResult<AccessTokenResult>(result);
        }
    }
```

微信公众平台规定了 access_token 有效期为 7200 秒，重复获取将导致上次获取的 access_token 失效，同时 access_tokenAPI 每天的访问次数被限制为 2000 次。如果每次调用高级接口都去请求 access_tokenAPI 获取 access_token，在调用频繁的情况下将很快达到 access_tokenAPI 每天访问限制，导致后续访问全部被微信服务器拒接。

因此，我们需要在程序中保存获取到的 access_token，每次访问高级接口先判断 access_token 是否已超过 7200 秒，如果没有超过有效期，则直接获取已保存的 access_token 调用高级接口。如果超过有效期，则重新获取一次 access_token，调用高级接口。

AccessTokenContainer 类实现了对 access_token 的自动管理，将上述逻辑封装在该类中，每次访问高级接口只需要调用 TryGetToken 方法获取 access_token 即可，无需关心 access_token 的有效期及 access_tokenAPI 的访问限制。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Senparc.Weixin.MP.Entities;
using Senparc.Weixin.MP.Helpers;

namespace Senparc.Weixin.MP.CommonAPIs
{
    /// <summary>
    /// AccessToken 容器
    /// </summary>
    class AccessTokenBag
    {
        /// <summary>
        /// 第三方用户唯一凭证
        /// </summary>
        public string AppId { get; set; }
        /// <summary>
        /// 第三方用户唯一凭证密钥
        /// </summary>
        public string AppSecret { get; set; }
        /// <summary>
        /// 凭证有效时间，单位：秒
        /// </summary>
        public DateTime ExpireTime { get; set; }
        /// <summary>
        /// 获取到的凭证
        /// </summary>
        public AccessTokenResult AccessTokenResult { get; set; }
    }
    /// <summary>
    /// 通用接口 AccessToken 容器，用于自动管理 AccessToken，过期会重新获取
    /// </summary>
    public class AccessTokenContainer
    {
        /// <summary>
```

```
        /// 保存不同公众号的AccessToken
        /// </summary>
        private static Dictionary<string, AccessTokenBag> AccessTokenCollection
=new Dictionary<string, AccessTokenBag>(StringComparer.OrdinalIgnoreCase);
        /// <summary>
        /// 使用完整的应用凭证获取 Token，不存在将自动注册
        /// </summary>
        /// <param name="appId"></param>
        /// <param name="appSecret"></param>
        /// <param name="getNewToken"></param>
        /// <returns></returns>
        public static string TryGetToken(string appId, string appSecret,
bool getNewToken = false)
        {
            if (!CheckRegistered(appId) || getNewToken)
            {
                Register(appId, appSecret);
            }
            return GetToken(appId);
        }
        /// <summary>
        /// 注册应用凭证信息，此操作只是注册，不会马上获取 Token，并将清空之前的 Token
        /// </summary>
        /// <param name="appId"></param>
        /// <param name="appSecret"></param>
        private static void Register(string appId, string appSecret)
        {
            AccessTokenCollection[appId] = new AccessTokenBag()
            {
                AppId = appId,
                AppSecret = appSecret,
                ExpireTime = DateTime.MinValue,
                AccessTokenResult = new AccessTokenResult()
            };
        }
        /// <summary>
        /// 检查是否已经注册
        /// </summary>
        /// <param name="appId"></param>
        /// <returns></returns>
        private static bool CheckRegistered(string appId)
```

```
        {
            return AccessTokenCollection.ContainsKey(appId);
        }
        /// <summary>
        /// 获取可用 Token
        /// </summary>
        /// <param name="appId"></param>
        /// <param name="getNewToken">是否强制重新获取新的 Token</param>
        /// <returns></returns>
        private static string GetToken(string appId, bool getNewToken = false)
        {
            return GetTokenResult(appId, getNewToken).access_token;
        }
        /// <summary>
        /// 获取可用 Token
        /// </summary>
        /// <param name="appId"></param>
        /// <param name="getNewToken">是否强制重新获取新的 Token</param>
        /// <returns></returns>
        private static AccessTokenResult GetTokenResult(string appId, bool
getNewToken = false)
        {
            if (!AccessTokenCollection.ContainsKey(appId))
            {
                throw new WeixinException("此 appId 尚未注册，请先使用
AccessTokenContainer.Register 完成注册（全局执行一次即可）！");
            }
            //从集合中获取 access_token
            var accessTokenBag = AccessTokenCollection[appId];
            if (getNewToken || accessTokenBag.ExpireTime <= DateTime.Now)
            {
                //如果集合中保存的 access_token 已过期，重新获取 accessTokenBag.
AccessTokenResult = Token.GetToken(accessTokenBag.AppId, accessTokenBag.
AppSecret);
                accessTokenBag.ExpireTime                                    =
DateTime.Now.AddSeconds(accessTokenBag.AccessTokenResult.expires_in);
            }
            return accessTokenBag.AccessTokenResult;
        }
    }
}
```

5.3　自定义菜单接口

微信公众平台的消息交互机制需要微信个人用户手工输入消息才能得到消息回复。由于手机屏幕的限制，在手机上输入文字并不太方便，一般我们使用手机，都是通过在手机屏幕上单击按钮来进行操作的。为了微信个人用户在访问微信公众号时获取更佳的用户体验，微信开发团队为微信公众平台增加了自定义菜单的功能。

5.3.1　自定义菜单简介

自定义菜单能够帮助公众号丰富界面，让用户更好更快地理解公众号的功能。开启自定义菜单后，公众号界面如图 5-3 所示。

开启自定义菜单后，微信个人用户只需要单击自定义菜单的不同菜单项，就能实现消息交互。

目前自定义菜单最多包括 3 个一级菜单，每个一级菜单最多包含 5 个二级菜单。一级菜单最多 4 个汉字，二级菜单最多 7 个汉字，多出来的部分将会以“...”代替。

创建自定义菜单后，由于微信客户端缓存，需要 24 小时微信客户端才会展现出来。测试时可以尝试取消关注公众账号后再次关注，这样可以立即看到创建后的效果。



图 5-3

5.3.2 自定义菜单数据结构

微信公众平台的自定义菜单数据是以 JSON 格式进行传输的，下面是一个完整的自定义菜单数据的例子：

```
{
    "button":[
    {
        "type":"click",
        "name":"今日歌曲",
        "key":"V1001_TODAY_MUSIC"
    },
    {
        "type":"click",
        "name":"歌手简介",
        "key":"V1001_TODAY_SINGER"
    },
    {
        "name":"菜单",
        "sub_button":[
        {
            "type":"view",
            "name":"搜索",
            "url":"http://www.soso.com/"
        },
        {
            "type":"view",
            "name":"视频",
            "url":"http://v.qq.com/"
        },
        {
            "type":"click",
            "name":"赞一下我们",
            "key":"V1001_GOOD"
        }]
    }]
}
```

自定义菜单数据中各参数的详细描述见表 5-1。目前自定义菜单接口可实现两种类型按钮：

1. click

用户单击 click 类型按钮后，微信服务器会通过消息接口推送消息类型为 event 的结构给开发者（参考消息接口指南），并且带上按钮中开发者填写的 key 值，开发者可以通过自定义的 key 值与用户进行交互。

2. view

用户单击 view 类型按钮后，微信客户端将会打开开发者在按钮中填写的 url 值（即网页链接），达到打开网页的目的，建议与网页授权获取用户基本信息接口结合，获得用户的个人信息。

表 5-1

参　数	是否必须	说　明
button	是	一级菜单数组，个数应为 1~3 个
sub_button	否	二级菜单数组，个数应为 1~5 个
type	是	菜单的响应动作类型，目前有 click、view 两种类型
name	是	菜单标题，不超过 16 个字节，子菜单不超过 40 个字节
key	click 类型必须	菜单 key 值，用于消息接口推送，不超过 128 字节
url	view 类型必须	网页链接，用户单击菜单可打开链接，不超过 256 字节

5.3.3　自定义菜单数据实体

首先建立一个继承于 WxJsonResult 的 JSON 数据实体类 GetMenuResultJson，用于对微信服务器发来的自定义菜单 JSON 数据格式进行反序列化操作。

```
/// <summary>
/// 获取菜单时候的完整结构，用于接收微信服务器返回的 JSON 信息
/// </summary>
public class GetMenuResultJson : WxJsonResult
{
    public MenuFull_ButtonGroup menu { get; set; }
}
public class MenuFull_ButtonGroup
{
    public List<MenuFull_RootButton> button { get; set; }
}
```

```
public class MenuFull_RootButton
{
    public string type { get; set; }
    public string key { get; set; }
    public string name { get; set; }
    public string url { get; set; }
    public List<MenuFull_RootButton> sub_button { get; set; }
}
```

同时，建立供C#语言内部使用的自定义菜单数据实体ButtonGroup，各菜单项的数据实体基类BaseButton，继承于BaseButton的单个菜单数据实体基类SingleButton，继承于BaseButton的子菜单数据实体SubButton，以及继承于SingleButton的click类型菜单和view类型菜单的数据实体SingleClickButton、SingleViewButton。

以上述结构建立的自定义菜单数据实体ButtonGroup可以直接序列化为发送给微信服务器的自定义菜单JSON数据格式，减少了ButtonGroup到GetMenuResultJson的转换操作。

```
public interface IBaseButton
{
    string name { get; set; }
}
/// <summary>
/// 所有按钮基类
/// </summary>
public class BaseButton : IBaseButton
{
    /// <summary>
    /// 按钮描述，即按钮名字，不超过16个字节，子菜单不超过40个字节
    /// </summary>
    public string name { get; set; }
}
/// <summary>
/// 所有单击按钮的基类（view，click等）
/// </summary>
public abstract class SingleButton : BaseButton, IBaseButton
{
    /// <summary>
    /// 按钮类型（click或view）
    /// </summary>
```

```
        public string type { get; set; }
        public SingleButton(string theType)
        {
            type = theType;
        }
    }
    /// <summary>
    /// 单个 click 按键
    /// </summary>
    public class SingleClickButton : SingleButton
    {
        /// <summary>
        /// 类型为 click 时必须
        /// 按钮 key 值，用于消息接口（event 类型）推送，不超过 128 字节
        /// </summary>
        public string key { get; set; }
        public SingleClickButton()
            : base(ButtonType.click.ToString())
        {
        }
    }
    /// <summary>
    /// 单个 view 按键
    /// </summary>
    public class SingleViewButton : SingleButton
    {
        /// <summary>
        /// 类型为 view 时必须
        /// 网页链接，用户单击按钮可打开链接，不超过 256 字节
        /// </summary>
        public string url { get; set; }
        public SingleViewButton()
            : base(ButtonType.view.ToString())
        {
        }
    }
    /// <summary>
    /// 子菜单
    /// </summary>
    public class SubButton : BaseButton, IBaseButton
    {
```

```
        /// <summary>
        /// 子按钮数组，按钮个数应为1~5个
        /// </summary>
        public List<SingleButton> sub_button { get; set; }
        public SubButton()
        {
            sub_button = new List<SingleButton>();
        }
        public SubButton(string name):this()
        {
            base.name = name;
        }
    }
    /// <summary>
    /// 整个按钮设置（可以直接用ButtonGroup实例返回JSON对象）
    /// </summary>
    public class ButtonGroup
    {
        /// <summary>
        /// 按钮数组，按钮个数应为1~3个
        /// </summary>
        public List<BaseButton> button { get; set; }
        public ButtonGroup()
        {
            button = new List<BaseButton>();
        }
    }
    /// <summary>
    /// GetMenu的返回结果实体
    /// </summary>
    public class GetMenuResult
    {
        public ButtonGroup menu { get; set; }
        public GetMenuResult()
        {
            menu = new ButtonGroup();
        }
    }
```

5.3.4　自定义菜单接口封装

微信公众平台为自定义菜单的操作开放了三个接口：

1. 自定义菜单创建接口

接口地址：https://api.weixin.qq.com/cgi-bin/menu/create?access_token= ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

2. 自定义菜单查询接口

接口地址：https://api.weixin.qq.com/cgi-bin/menu/get?access_token=ACCESS_TOKEN

HTTP 请求方式：GET

3. 自定义菜单删除接口

接口地址：https://api.weixin.qq.com/cgi-bin/menu/delete?access_token= ACCESS_TOKEN

HTTP 请求方式：GET

为了便于识别自定义菜单接口产生的错误，我们首先定义一个自定义错误类型 WeixinMenuException。所有自定义菜单接口操作中发生的异常，都将抛出一个 WeixinMenuException 错误。

```
/// <summary>
/// 微信自定义菜单接口类型错误
/// </summary>
public class WeixinMenuException : WeixinException
{
    public WeixinMenuException(string message)
        : base(message, null)
    {
    }
    public WeixinMenuException(string message, Exception inner)
        : base(message, inner)
```

```
        {
        }
    }
```

接下来新建一个 Meun 类，具体实现微信公众平台开放的三个自定义菜单操作接口，包括自定义菜单创建接口 CreateMenu，自定义菜单查询接口 GetMenu 及自定义菜单删除接口 DeleteMenu 等。

首次创建自定义菜单及以后的每次更新自定义菜单，可以直接调用自定义菜单创建接口 CreateMenu，不需要先删除再创建。

```
    using System;
    using System.Collections;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Net;
    using System.Text;
    using System.Web.Script.Serialization;
    using Senparc.Weixin.MP.Entities;
    using Senparc.Weixin.MP.Entities.Menu;
    using Senparc.Weixin.MP.Exceptions;
    using Senparc.Weixin.MP.Helpers;

    namespace Senparc.Weixin.MP.CommonAPIs
    {
        /// <summary>
        /// 微信公众平台自定义菜单操作类
        /// </summary>
        public class Meun
        {
            /// <summary>
            /// 创建菜单
            /// </summary>
            /// <param name="accessToken"></param>
            /// <param name="buttonData">菜单内容</param>
            /// <returns></returns>
            public static WxJsonResult CreateMenu(string accessToken, ButtonGroup
buttonData)
            {
                //微信公众平台创建自定义菜单接口地址
```

```
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/menu/create?
access_token={0}";
            return ApiHelper.Post(accessToken, urlFormat, buttonData);
        }
        #region GetMenu
        /// <summary>
        /// 获取当前菜单，如果菜单不存在，则返回 null
        /// </summary>
        /// <param name="accessToken"></param>
        /// <returns></returns>
        public static GetMenuResult GetMenu(string accessToken)
        {
            //微信公众平台获取自定义菜单接口地址
            var url = string.Format("https://api.weixin.qq.com/cgi-bin/menu/
get?access_token={0}", accessToken);
            var jsonString = HttpUtility.RequestUtility.HttpGet(url, Encoding.
UTF8);
            GetMenuResult finalResult;
            JavaScriptSerializer js = new JavaScriptSerializer();
            try
            {
                //获取自定义菜单 JSON 结构
                var jsonResult = js.Deserialize<GetMenuResultJson> (jsonString);
                if (jsonResult.menu == null || jsonResult.menu.button.Count
== 0)
                {
                    throw new WeixinException(jsonResult.errmsg);
                }
                //将 JSON 结构自定义菜单转换为自定义菜单实体
                finalResult = GetMenuFromJsonResult(jsonResult);
            }
            catch (WeixinException)
            {
                finalResult = null;
            }
            return finalResult;
        }
        /// <summary>
        /// 根据微信返回的 JSON 数据得到可用的 GetMenuResult 结果
        /// 将 JSON 结构自定义菜单转换为自定义菜单实体
        /// </summary>
```

```
        /// <param name="resultFull"></param>
        /// <returns></returns>
        private static GetMenuResult GetMenuFromJsonResult (GetMenuResultJson
resultFull)
        {
            GetMenuResult result = null;
            try
            {
                ButtonGroup bg = new ButtonGroup();
                //循环遍历 JSON 结构
                foreach (var rootButton in resultFull.menu.button)
                {
                    if (rootButton.name == null)
                    {
                        continue;//没有设置一级菜单
                    }
                    //可用二级菜单按钮数量
                    var availableSubButton = rootButton.sub_button.Count(z
=> !string.IsNullOrEmpty(z.name));
                    //一级菜单格式转换
                    if (availableSubButton == 0)
                    {
                        //按钮格式校验
                        if (rootButton.type == null ||
                            (rootButton.type.Equals("CLICK", StringComparison.
OrdinalIgnoreCase)
                            && string.IsNullOrEmpty(rootButton.key)))
                        {
                            throw new WeixinMenuException("单击按钮的 key 不能为
空! ");
                        }
                        if (rootButton.type.Equals("CLICK", StringComparison.
OrdinalIgnoreCase))
                        {
                            //底部单击按钮
                            bg.button.Add(new SingleClickButton()
                            {
                                name = rootButton.name,
                                key = rootButton.key,
                                type = rootButton.type
                            });
```

```
                    }
                    else
                    {
                        //底部 URL 按钮
                        bg.button.Add(new SingleViewButton()
                        {
                            name = rootButton.name,
                            url = rootButton.url,
                            type = rootButton.type
                        });
                    }
                }
                else if (availableSubButton < 1)
                {
                    throw new WeixinMenuException("子菜单至少需要填写 1 个!");
                }
                //二级菜单格式转换
                else
                {
                    //底部二级菜单
                    var subButton = new SubButton(rootButton.name);
                    bg.button.Add(subButton);
                    foreach (var subSubButton in rootButton.sub_button)
                    {
                        if (subSubButton.name == null)
                        {
                            continue; //没有设置菜单
                        }
                        //按钮格式校验
                        if (subSubButton.type.Equals("CLICK", StringComparison.
OrdinalIgnoreCase)
                            && string.IsNullOrEmpty(subSubButton.key))
                        {
                            throw new WeixinMenuException("单击按钮的 key 不能
为空! ");
                        }
                        if (subSubButton.type.Equals("CLICK", StringComparison.
OrdinalIgnoreCase))
                        {
                            //底部单击按钮
                            subButton.sub_button.Add(new SingleClickButton()
```

```
                        {
                            name = subSubButton.name,
                            key = subSubButton.key,
                            type = subSubButton.type
                        });
                    }
                    else
                    {
                        //底部URL按钮
                        subButton.sub_button.Add(new SingleViewButton()
                        {
                            name = subSubButton.name,
                            url = subSubButton.url,
                            type = subSubButton.type
                        });
                    }
                }
            }
        }
        if (bg.button.Count < 1)
        {
            throw new WeixinMenuException("一级菜单按钮至少为1个！");
        }
        //保存到自定义菜单实体
        result = new GetMenuResult()
        {
            menu = bg
        };
    }
    catch (Exception ex)
    {
        throw new WeixinMenuException(ex.Message, ex);
    }
    return result;
}
#endregion
/// <summary>
/// 删除菜单
/// </summary>
/// <param name="accessToken"></param>
/// <returns></returns>
```

```
        public static WxJsonResult DeleteMenu(string accessToken)
        {
            //微信公众平台删除自定义菜单接口地址
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/menu/delete?
access_token={0}";
            var result = ApiHelper.Get(accessToken, urlFormat);
            return result;
        }
    }
}
```

5.4　多媒体文件接口

微信个人用户向微信公众号发送的多媒体消息，如图片消息、语音消息、视频消息等，会首先保存在微信服务器中。微信服务器将微信用户的多媒体消息发送给微信公众号时，只发送了保存在微信服务器中的多媒体文件的 MediaId。微信公众号需要调用微信公众平台的多媒体文件接口，才能获取多媒体文件的具体内容。

同样，微信公众号向微信个人用户发送多媒体消息时，也需要先调用微信公众平台的多媒体文件接口，将多媒体文件上传到微信服务器得到 MediaId，再将包含 MediaId 的多媒体消息发送给微信个人用户。

5.4.1　多媒体文件接口简介

1. 上传多媒体文件接口

微信公众号可调用该接口来上传图片、语音、视频等文件到微信服务器，上传后服务器会返回对应的 media_id，公众号此后可根据该 media_id 来获取多媒体。

接口地址：http://file.api.weixin.qq.com/cgi-bin/media/upload?

access_token=ACCESS_TOKEN&type=TYPE

HTTP 请求方式：POST/FORM

请求参数描述：见表 5-2。

返回结果：在正确情况下将返回JSON数据包，返回结果的参数描述见表5-3。

表 5-2

参　数	是否必须	说　明
access_token	是	调用接口凭证
type	是	媒体文件类型，分别有图片（image）、语音（voice）、视频（video）和缩略图（thumb）
media	是	form-data 中媒体文件标识，有 filename、filelength、content-type 等信息

表 5-3

参　数	描　述
type	媒体文件类型，分别有图片（image）、语音（voice）、视频（video）和缩略图（thumb，主要用于视频与音乐格式的缩略图）
media_id	媒体文件上传后，获取时的唯一标识
created_at	媒体文件上传时间戳

上传的多媒体文件有格式和大小限制，如下：

（1）图片（image）: 128K，支持 JPG 格式

（2）语音（voice）：256K，播放长度不超过 60s，支持 AMR\MP3 格式

（3）视频（video）：1MB，支持 MP4 格式

（4）缩略图（thumb）：64KB，支持 JPG 格式

媒体文件在后台保存时间为 3 天，即 3 天后 media_id 失效。

2. 下载多媒体文件

公众号可调用该接口来获取多媒体文件。

接口地址：http://file.api.weixin.qq.com/cgi-bin/media/get?

access_token=ACCESS_TOKEN&media_id=MEDIA_ID

HTTP 请求方式: GET

请求参数描述：见表 5-4。

表 5-4

参　数	是否必须	说　明
access_token	是	调用接口凭证
media_id	是	媒体文件 ID

视频文件不支持下载。

5.4.2　上传下载文件

为了能访问多媒体文件接口，我们首先需要在 HTTP 访问工具类 RequestUtility 中添加两个方法，Download 方法实现 HTTP 下载文件，Upload 方法实现 HTTP 上传文件。

HTTP 下载文件，采用 WebClient 即可简单实现。

```
/// <summary>
/// 使用 GET 方法下载文件
/// </summary>
/// <param name="url"></param>
/// <param name="stream">下载的文件流信息</param>
public static void Download(string url, Stream stream)
{
    WebClient wc = new WebClient();
    var data = wc.DownloadData(url);
    foreach (var b in data)
    {
        stream.WriteByte(b);
    }
}
```

HTTP 上传文件首先定义一个读取文件帮助类 FileHelper，用于从文件系统中读取文件。

```
/// <summary>
/// 读取文件帮助类
/// </summary>
public class FileHelper
{
    /// <summary>
    /// 根据完整文件路径获取 FileStream
```

```
        /// </summary>
        /// <param name="fileName"></param>
        /// <returns></returns>
        public static FileStream GetFileStream(string fileName)
        {
            FileStream fileStream = null;
            if (!string.IsNullOrEmpty(fileName) && File.Exists(fileName))
            {
                fileStream = new FileStream(fileName, FileMode.Open);
            }
            return fileStream;
        }
    }
```

HTTP 上传文件采用 HttpWebRequest/ HttpWebResponse 模拟 Form 表单文件上传的方式实现。

```
    /// <summary>
    /// 使用 POST 方法上传文件
    /// </summary>
    /// <param name="url"></param>
    /// <param name="fileDictionary">需要上传的文件，key：对应要上传的 name，value：本地文件名</param>
    /// <returns></returns>
    public static string Upload(string url, Dictionary<string, string> fileDictionary, Encoding encoding = null)
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        request.Method = "POST";
        Stream postStream = new MemoryStream();
        #region 处理 Form 表单文件上传
        var formUploadFile = fileDictionary != null && fileDictionary.Count > 0;//是否用 Form 上传文件
        if (formUploadFile)
        {
            //通过表单上传文件
            string boundary = "----" + DateTime.Now.Ticks.ToString("x");
            string formdataTemplate = "\r\n--" + boundary + "\r\nContent-Disposition: form-data; name=\"{0}\"; filename=\"{1}\"\r\nContent-Type: application/ octet-stream\r\n\r\n";
            foreach (var file in fileDictionary)
            {
```

```
                try
                {
                    var fileName = file.Value;
                    //准备文件流
                    using (var fileStream = FileHelper.GetFileStream (fileName))
                    {
                        var formdata = string.Format(formdataTemplate, file.
Key, fileName /*Path.GetFileName(fileName)*/);
                        var formdataBytes = Encoding.ASCII.GetBytes (postStream.
Length == 0 ? formdata.Substring(2, formdata.Length - 2) : formdata);//第一行不需要
换行
                        postStream.Write(formdataBytes, 0, formdataBytes.
Length);
                        //写入文件
                        byte[] buffer = new byte[1024];
                        int bytesRead = 0;
                        while ((bytesRead = fileStream.Read(buffer, 0, buffer.
Length)) != 0)
                        {
                            postStream.Write(buffer, 0, bytesRead);
                        }
                    }
                }
                catch (Exception ex)
                {
                    throw ex;
                }
            }
            //结尾
            var footer = Encoding.ASCII.GetBytes("\r\n--" + boundary +
"--\r\n");
            postStream.Write(footer, 0, footer.Length);
            request.ContentType = string.Format("multipart/form-data; boundary=
{0}", boundary);
        }
        #endregion
        request.ContentLength = postStream != null ? postStream.Length : 0;
        request.Accept                                                    =
"text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
";
        request.KeepAlive = true;
```

```
        request.UserAgent = "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/
537.36 (KHTML, like Gecko) Chrome/31.0.1650.57 Safari/537.36";
        #region 输入二进制流
        if (postStream != null)
        {
            postStream.Position = 0;
            //直接写入流
            Stream requestStream = request.GetRequestStream();
            byte[] buffer = new byte[1024];
            int bytesRead = 0;
            while ((bytesRead = postStream.Read(buffer, 0, buffer.Length)) != 0)
            {
                requestStream.Write(buffer, 0, bytesRead);
            }
            postStream.Close();//关闭文件访问
        }
        #endregion
        HttpWebResponse response = (HttpWebResponse)request.GetResponse();
        using (Stream responseStream = response.GetResponseStream())
        {
            using (StreamReader myStreamReader = new StreamReader(responseStream,
encoding ?? Encoding.GetEncoding("utf-8")))
            {
                string retString = myStreamReader.ReadToEnd();
                return retString;
            }
        }
    }
```

5.4.3 多媒体文件接口封装

首先，定义一个多媒体文件接口支持的多媒体文件类型枚举 UploadMediaFileType，以及一个用于保存上传文件返回结果的 JSON 数据实体 UploadResultJson。

```
    /// <summary>
    /// 上传媒体文件类型
    /// </summary>
    public enum UploadMediaFileType
    {
        /// <summary>
        /// 图片：128K，支持 JPG 格式
```

```
        /// </summary>
        image,
        /// <summary>
        /// 语音：256K，播放长度不超过 60s，支持 AMR\MP3 格式
        /// </summary>
        voice,
        /// <summary>
        /// 视频：1MB，支持 MP4 格式
        /// </summary>
        video,
        /// <summary>
        /// thumb：64KB，支持 JPG 格式
        /// </summary>
        thumb
    }
    /// <summary>
    /// 上传媒体文件返回结果
    /// </summary>
    public class UploadResultJson : WxJsonResult
    {
        public UploadMediaFileType type { get; set; }
        public string media_id { get; set; }
        public long created_at { get; set; }
    }
```

接下来在接口访问方法类 ApiHelper 中添加上传、下载文件的接口操作方法 Download 与 Upload。

```
    /// <summary>
    /// 下载文件
    /// </summary>
    /// <param name="urlFormat">API 接口地址格式</param>
    /// <param name="stream">下载的文件流信息</param>
    /// <param name="querys">需要传递的参数</param>
    public static void Download(string urlFormat, Stream stream, params string[]
querys)
    {
        RequestUtility.Download(string.Format(urlFormat, querys), stream);
    }
    /// <summary>
    /// 上传文件
    /// </summary>
```

```
    /// <param name="urlFormat">API 接口地址格式</param>
    /// <param name="accessToken">微信公众号访问授权 AccessToken</param>
    /// <param name="fileDictionary">需要上传的文件，key：对应要上传的 name，value：
本地文件名</param>
    /// <param name="querys">除了 AccessToken 还需要传递的其他参数</param>
    public static T Upload<T>(string accessToken, string urlFormat, Dictionary
<string, string> fileDictionary, params string[] querys)
    {
        var url = GetApiUrl(urlFormat, accessToken, querys);
        string returnText = HttpUtility.RequestUtility.Upload(url,
fileDictionary);
        var result = GetResult<T>(returnText);
        return result;
    }
```

最后新建一个类 Media，调用 ApiHelper 中的 Download、Upload 方法，来实现多媒体文件接口。

```
    using Senparc.Weixin.MP.Helpers;
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Text;

    namespace Senparc.Weixin.MP.AdvancedAPIs
    {
        /// <summary>
        /// 微信公众号多媒体文件接口
        /// </summary>
        public static class Media
        {
            /// <summary>
            /// 上传媒体文件
            /// </summary>
            /// <param name="accessToken"></param>
            /// <param name="type">上传文件类型</param>
            /// <param name="file">上传文件物理路径</param>
            /// <returns></returns>
            public static UploadResultJson Upload(string accessToken,
UploadMediaFileType type, string file)
            {
```

```
            //微信公众号上传媒体文件接口地址
            var urlFormat = "http://file.api.weixin.qq.com/cgi-bin/media/
upload?access_token={0}&type={1}";
            var fileDictionary = new Dictionary<string, string>();
            fileDictionary["media"] = file;
            return ApiHelper.Upload<UploadResultJson>(accessToken, urlFormat,
fileDictionary, type.ToString());
        }
        /// <summary>
        /// 下载媒体文件
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="mediaId">媒体文件 ID</param>
        /// <param name="stream">下载结果</param>
        public static void Get(string accessToken, string mediaId, Stream
stream)
        {
            //微信公众号下载媒体文件接口地址
            var urlFormat = "http://file.api.weixin.qq.com/cgi-bin/media/get?
access_token={0}&media_id={1}";
            ApiHelper.Download(urlFormat, stream, accessToken, mediaId);
        }
    }
}
```

通过对代码结构进行上述封装，在具体使用时，只需要调用 Media 中的 Upload 与 GET 方法，即可实现对多媒体文件接口的操作，开发者不需要知道接口调用地址，也无需关心接口具体实现方式。在根据具体业务场景，进行微信公众号的多媒体文件应用开发时，使用 Media 类，将大大简化开发难度与复杂度。

5.5　用户管理接口

用户管理接口由用户信息接口与用户分组接口等两大类组成。用户信息接口包括获取关注者列表接口与获取用户基本信息接口。用户分组接口包括创建分组接口、查询所有分组接口、查询用户所在分组接口、修改分组名接口及移动用户分组接口。

5.5.1 用户信息接口简介

1. 获取关注者列表接口

微信公众号可通过该接口来获取账号的关注者列表，关注者列表由一串OpenId（加密后的微信号，每个用户对每个公众号的OpenId是唯一的）组成。一次拉取调用最多10 000个关注者的OpenId。

接口地址：https://api.weixin.qq.com/cgi-bin/user/get?

access_token=ACCESS_TOKEN&next_openid=NEXT_OPENID

HTTP请求方式：GET（请使用HTTPS协议）

请求参数描述：见表5-5。

返回结果：在正确情况下将返回JSON数据包，返回结果参数描述见表5-6。

表5-5

参　数	是否必须	说　明
access_token	是	调用接口凭证
next_openid	是	第一个拉取的OpenId不填，默认从头开始拉取

表5-6

参　数	说　明
total	关注该公众账号的总用户数
count	拉取的OpenId个数，最大值为10 000
data	列表数据，OpenId的列表
next_openid	拉取列表的后一个用户的OpenId

当公众号关注者数量超过10 000时，将上一次调用得到的返回中的next_openid值，作为下一次调用中的next_openid值，通过多次拉取列表的方式获取所有关注者。

2. 获取用户基本信息接口

在关注者与公众号产生消息交互后，公众号可获得关注者的OpenId。公众号可通过本接口来根据OpenId获取用户基本信息，包括昵称、头像、性别、所在城

市、语言和关注时间。

接口地址：https://api.weixin.qq.com/cgi-bin/user/info?

access_token=ACCESS_TOKEN&openid=OPENID&lang=zh_CN

HTTP 请求方式：GET（请使用 HTTPS 协议）

请求参数描述：见表 5-7。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-8。

表 5-7

参　数	是否必须	说　明
access_token	是	调用接口凭证
openid	是	普通用户的标识，对当前公众号唯一
lang	否	返回国家地区语言版本，zh_CN 简体、zh_TW 繁体、en 英语

表 5-8 为获取用户基本信息接口返回结果参数描述。

表 5-8

参　数	说　明
subscribe	用户是否订阅该公众号标识，值为 0 时，代表此用户没有关注该公众号，拉取不到其余信息
openid	用户的标识，对当前公众号唯一
nickname	用户的昵称
sex	用户的性别，值为 1 时是男性，值为 2 时是女性，值为 0 时是未知
city	用户所在城市
country	用户所在国家
province	用户所在省份
language	用户的语言，简体中文为 zh_CN
headimgurl	用户头像，最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0 代表 640×640 正方形头像），用户没有头像时该项为空
subscribe_time	用户关注时间，为时间戳。如果用户曾多次关注，则取最后关注时间

5.5.2 用户信息接口封装

根据接口描述，首先定义关注者列表 JSON 数据实体 OpenIdResultJson 及用户信息 JSON 数据实体 UserInfoJson。

```
    /// <summary>
    /// 用户信息返回结果
    /// </summary>
    public class UserInfoJson
    {
        /// <summary>
        /// 用户是否订阅该公众号标识，值为0时，代表此用户没有关注该公众号，拉取不到其余信息
        /// </summary>
        public int subscribe { get; set; }
        /// <summary>
        /// 用户的标识，对当前公众号唯一
        /// </summary>
        public string openid { get; set; }
        /// <summary>
        /// 用户的昵称
        /// </summary>
        public string nickname { get; set; }
        /// <summary>
        /// 用户的性别，值为1时是男性，值为2时是女性，值为0时是未知
        /// </summary>
        public int sex { get; set; }
        /// <summary>
        /// 用户的语言
        /// </summary>
        public string language { get; set; }
        /// <summary>
        /// 用户所在城市
        /// </summary>
        public string city { get; set; }
        /// <summary>
        /// 用户所在省份
        /// </summary>
        public string province { get; set; }
        /// <summary>
        /// 用户所在国家
```

```
        /// </summary>
        public string country { get; set; }
        /// <summary>
        /// 用户头像，最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0
代表 640×640 正方形头像），用户没有头像时该项为空
        /// </summary>
        public string headimgurl { get; set; }
        /// <summary>
        /// 用户关注时间，为时间戳。如果用户曾多次关注，则取最后关注时间
        /// </summary>
        public long subscribe_time { get; set; }
    }
    /// <summary>
    /// 关注者列表返回结果
    /// </summary>
    public class OpenIdResultJson : WxJsonResult
    {
        /// <summary>
        /// 关注该公众账号的总用户数
        /// </summary>
      public int total { get; set; }
        /// <summary>
      /// 拉取的 OpenId 个数，最大值为 10 000
        /// </summary>
      public int count { get; set; }
        /// <summary>
      /// OpenId 的列表
        /// </summary>
      public OpenIdResultJson_Data data { get; set; }
        /// <summary>
      /// 拉取列表的后一个用户的 OpenId
        /// </summary>
      public string next_openid { get; set; }
    }
    /// <summary>
    /// 关注者 OpenId 列表
    /// </summary>
    public class OpenIdResultJson_Data
    {
        public List<string> openid { get; set; }
    }
```

然后新建一个类 User，调用 ApiHelper 中的 GET 方法，来实现获取关注者列表接口 List，以及获取用户基本信息接口 Info。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Senparc.Weixin.MP.Helpers;

namespace Senparc.Weixin.MP.AdvancedAPIs
{
    /// <summary>
    /// 用户接口
    /// </summary>
    public static class User
    {
        /// <summary>
        /// 获取用户信息
        /// </summary>
        /// <param name="accessToken">调用接口凭证</param>
        /// <param name="openId">普通用户的标识，对当前公众号唯一</param>
        /// <param name="lang">返回国家地区语言版本，zh_CN 简体、zh_TW 繁体、en
英语</param>
        /// <returns></returns>
        public static UserInfoJson Info(string accessToken, string openId,
Language lang = Language.zh_CN)
        {
            //微信公众平台获取用户信息接口地址
            string urlFormat = "https://api.weixin.qq.com/cgi-bin/user/info?
access_token={0}&openid={1}&lang={2}";
            return ApiHelper.Get<UserInfoJson>(accessToken, urlFormat, openId,
lang.ToString());
            //错误时微信会返回错误码等信息，JSON 数据包示例如下（该示例为 AppID 无效错
误）
            //{"errcode":40013,"errmsg":"invalid appid"}
        }
        /// <summary>
        /// 获取关注者 OpenId 信息
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="nextOpenId"></param>
```

```
        /// <returns></returns>
        public static OpenIdResultJson List(string accessToken, string
nextOpenId)
        {
            //微信公众平台获取关注者列表接口地址
            string urlFormat = "https://api.weixin.qq.com/cgi-bin/user/
get?access_token={0}";
            if (!string.IsNullOrEmpty(nextOpenId))
            {
                urlFormat += "&next_openid={1}";
                return ApiHelper.Get<OpenIdResultJson>(accessToken, urlFormat,
nextOpenId);
            }
            else
                return ApiHelper.Get<OpenIdResultJson>(accessToken, urlFormat);
        }
    }
```

5.5.3　用户分组接口简介

1. 创建分组接口

一个微信公众号，最多支持创建 500 个分组。

接口地址：https://api.weixin.qq.com/cgi-bin/groups/create?access_token=ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：JSON 数据包，参数描述见表 5-9。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-10。

表 5-9

参　数	说　明
access_token	调用接口凭证
name	分组名字（30 个字符以内）

表 5-10

参　数	说　明
id	分组 ID，由微信分配
name	分组名字，UTF8 编码

2. 查询所有分组接口

接口地址：https://api.weixin.qq.com/cgi-bin/groups/get?access_token=ACCESS_TOKEN

HTTP 请求方式：GET（请使用 HTTPS 协议）

请求参数描述：见表 5-11。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-12。

表 5-11

参　数	说　明
access_token	调用接口凭证

表 5-12

参　数	说　明
groups	公众平台分组信息列表
id	分组 ID，由微信分配
name	分组名字，UTF8 编码
count	分组内用户数量

3. 查询用户所在分组接口

接口地址：https://api.weixin.qq.com/cgi-bin/groups/getid?access_token=ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：JSON 数据包，参数描述见表 5-13。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-14。

表 5-13

参　数	说　明
access_token	调用接口凭证
openid	用户的 OpenId

表 5-14

参　数	说　明
groupid	用户所属的 groupid

4. 修改分组名接口

接口地址：https://api.weixin.qq.com/cgi-bin/groups/ update?

access_token=ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：JSON 数据包，参数描述见表 5-15。

返回结果：在正确情况下将返回 JSON 数据包{"errcode": 0, "errmsg": "ok"}。

表 5-15

参　数	说　明
access_token	调用接口凭证
id	分组 ID，由微信分配
name	分组名字（30 个字符以内）

5. 移动用户分组

接口地址：https://api.weixin.qq.com/cgi-bin/groups/ members /update?

access_token=ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：JSON 数据包，参数描述见表 5-16。

返回结果：在正确情况下将返回 JSON 数据包{"errcode": 0, "errmsg": "ok"}。

表 5-16

参　数	说　明
access_token	调用接口凭证
openid	用户唯一标识符
to_groupid	分组 ID

5.5.4 用户分组接口封装

根据接口描述，首先定义单个分组 JSON 数据实体 GroupsJson，分组列表 JSON 数据实体 GroupsJson_Group，创建分组结果 JSON 实体 CreateGroupResult，以及用户所在分组 JSON 数据实体 GetGroupIdResult。

```
/// <summary>
/// 用户分组列表
/// </summary>
public class GroupsJson : WxJsonResult
{
    public List<GroupsJson_Group> groups { get; set; }
}
/// <summary>
/// 单个用户分组
/// </summary>
public class GroupsJson_Group
{
    /// <summary>
    /// 分组 ID，由微信分配
    /// </summary>
    public int id { get; set; }
    /// <summary>
    /// 分组名字（30 个字符以内）
    /// </summary>
    public string name { get; set; }
    /// <summary>
    /// 分组内用户数量
    /// 此属性在 CreateGroupResult 的 JSON 数据中，创建结果中始终为 0
    /// </summary>
    public int count { get; set; }
}
```

```
/// <summary>
/// 创建分组返回结果
/// </summary>
public class CreateGroupResult : WxJsonResult
{
    public GroupsJson_Group group { get; set; }
}
/// <summary>
/// 查询单个用户所在分组返回结果
/// </summary>
public class GetGroupIdResult : WxJsonResult
{
    /// <summary>
    /// 用户所在的分组 ID，由微信分配
    /// </summary>
    public int groupid { get; set; }
}
```

然后新建一个类 Groups，调用 ApiHelper 中的 GET 与 POST 方法，来实现用创建分组接口 Create，查询所有分组接口 GET，查询用户所在分组接口 GetID，修改分组名接口 Update，以及移动用户分组接口 MemberUpdate。

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Web.Script.Serialization;
using Senparc.Weixin.MP.Entities;
using Senparc.Weixin.MP.Helpers;

namespace Senparc.Weixin.MP.AdvancedAPIs
{
    /// <summary>
    /// 用户分组接口
    /// </summary>
    public static class Groups
    {
        /// <summary>
        /// 创建分组
```

```
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="name">分组名称（30 个字符以内）</param>
        /// <returns></returns>
        public static CreateGroupResult Create(string accessToken, string
name)
        {
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/groups/create?
access_token={0}";
            var data = new
            {
                group = new
                {
                    name = name
                }
            };
            return ApiHelper.Post<CreateGroupResult>(accessToken, urlFormat,
data);
        }
        /// <summary>
        /// 获取分组列表
        /// </summary>
        /// <param name="accessToken"></param>
        /// <returns></returns>
        public static GroupsJson Get(string accessToken)
        {
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/groups/get?
access_token={0}";
            return ApiHelper.Get<GroupsJson>(accessToken, urlFormat);
        }
        /// <summary>
        /// 获取单个用户所属分组
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">用户 OpenId</param>
        /// <returns></returns>
        public static GetGroupIdResult GetId(string accessToken, string
openId)
        {
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/groups/getid?
access_token={0}";
```

```
            var data = new { openid = openId };
            return ApiHelper.Post<GetGroupIdResult>(accessToken, urlFormat,
data);
        }
        /// <summary>
        /// 修改分组名称
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="id">分组 ID</param>
        /// <param name="name">新的分组名称（30 个字符以内）</param>
        /// <returns></returns>
        public static WxJsonResult Update(string accessToken, int id, string
name)
        {
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/groups/update?
access_token={0}";
            var data = new
            {
                group = new
                {
                    id = id,
                    name = name
                }
            };
            return ApiHelper.Post(accessToken, urlFormat, data);
        }
        /// <summary>
        /// 移动单个用户到指定分组
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">用户 OpenId</param>
        /// <param name="toGroupId">要移动到的分组 ID</param>
        /// <returns></returns>
        public static WxJsonResult MemberUpdate(string accessToken, string
openId, int toGroupId)
        {
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/groups/members/
update?access_token={0}";
            var data = new
            {
                openid = openId,
```

```
                to_groupid = toGroupId
            };
            return ApiHelper.Post(accessToken, urlFormat, data);
        }
    }
}
```

5.6 客服接口

一般的微信公众平台消息交互，只有在普通微信用户向公众号发送消息后，公众号才能回复消息给普通微信用户。而客服接口，允许微信公众号主动发消息给普通微信用户。

5.6.1 客服接口简介

当微信个人用户主动发消息给微信公众号的时候，微信公众号在 48 小时内可以调用客服消息接口，通过 POST 一个 JSON 数据包来发送消息给微信个人用户，在 48 小时内不限制发送次数。该接口主要用于客服等有人工消息处理环节的功能，方便微信公众号为用户提供更加优质的服务。

目前客服接口支持的发送消息类型有以下几种。

接口地址：https://api.weixin.qq.com/cgi-bin/message/custom/send?

access_token=ACCESS_TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：不同类型消息的 JSON 数据包。

返回结果：在正确情况下将返回 JSON 数据包{"errcode": 0, "errmsg": "ok"}。

客服接口支持的发送消息类型共有 6 种。

1. 文本消息

文本消息的 JSON 数据包格式如下，各参数描述见表 5-17：

```
{
    "touser":"OPENID",
    "msgtype":"text",
    "text":
    {
        "content":"Hello World"
    }
}
```

表 5-17

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，text
content	是	文本消息内容

2. 图片消息

发送图片消息之前，需要先将图片通过多媒体文件上传接口上传到微信服务器，得到图片的 MediaId。

图片消息的 JSON 数据包格式如下，各参数描述见表 5-18：

```
{
    "touser":"OPENID",
    "msgtype":"image",
    "image":
    {
      "media_id":"MEDIA_ID"
    }
}
```

表 5-18

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，image
media_id	是	发送的图片的媒体 Id

3. 语音消息

发送语音消息之前，需要先将语音文件通过多媒体文件上传接口上传到微信服务器，得到语音的 MediaId。

语音消息的 JSON 数据包格式如下，各参数描述见表 5-19：

```
{
   "touser":"OPENID",
   "msgtype":"voice",
   "voice":
   {
     "media_id":"MEDIA_ID"
   }
}
```

表 5-19

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，voice
media_id	是	发送的语音的媒体 Id

4. 视频消息

发送视频消息之前，需要先将视频文件通过多媒体文件上传接口上传到微信服务器，得到视频的 MediaId。

视频消息的 JSON 数据包格式如下，各参数描述见表 5-20：

```
{
   "touser":"OPENID",
   "msgtype":"video",
   "video":
   {
     "media_id":"MEDIA_ID",
     "title":"TITLE",
     "description":"DESCRIPTION"
   }
}
```

表 5-20

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，video
media_id	是	发送的视频的媒体 Id
title	否	视频消息的标题
description	否	视频消息的描述

5. 音乐消息

音乐消息的 JSON 数据包格式如下，各参数描述见表 5-21：

```
{
    "touser":"OPENID",
    "msgtype":"music",
    "music":
    {
      "title":"MUSIC_TITLE",
      "description":"MUSIC_DESCRIPTION",
      "musicurl":"MUSIC_URL",
      "hqmusicurl":"HQ_MUSIC_URL",
      "thumb_media_id":"THUMB_MEDIA_ID"
    }
}
```

表 5-21

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，music
title	否	音乐标题
description	否	音乐描述
musicurl	是	音乐链接
hqmusicurl	是	高品质音乐链接，Wi-Fi 环境优先使用该链接播放音乐
thumb_media_id	是	缩略图的媒体 Id

6. 图文消息

图文消息条数限制在 10 条以内，注意，如果图文数超过 10，则将会无响应。图文消息的 JSON 数据包格式如下，各参数描述见表 5-22：

```
{
   "touser":"OPENID",
   "msgtype":"news",
   "news":{
      "articles": [
       {
          "title":"Happy Day",
          "description":"Is Really A Happy Day",
          "url":"URL",
          "picurl":"PIC_URL"
       },
       {
          "title":"Happy Day",
          "description":"Is Really A Happy Day",
          "url":"URL",
          "picurl":"PIC_URL"
       }
       ]
   }
}
```

表 5-22

参　数	是否必须	说　明
access_token	是	调用接口凭证
touser	是	普通用户 OpenId
msgtype	是	消息类型，news
title	否	标题
description	否	描述
url	否	单击后跳转的链接
picurl	否	图文消息的图片链接，支持 JPG、PNG 格式，较好的效果为大图 640×320，小图 80×80

5.6.2　客服接口封装

根据接口描述，新建一个类 CustomerService，调用 ApiHelper 中的 POST 方法，来实现发送文本消息 SendText、发送图片消息 SendImage、发送语音消息 SendVoice、发送视频消息 SendVideo、发送音乐消息 SendMusic 和发送图文消息 SendNews。

其中 SendImage、SendVoice、SendVideo 需要先调用多媒体文件接口类 Media 的 Upload 方法，上传媒体文件到微信服务器并获取 MediaId。

SendNews 传入的消息为第 4 章消息框架中的公众号回复图文消息类 Article 的集合，图文消息条数不能超过 10 条。

```
    using System;
    using System.Collections.Generic;
    using System.IO;
    using System.Linq;
    using System.Net;
    using System.Security.Cryptography;
    using System.Text;
    using System.Web.Script.Serialization;
    using Senparc.Weixin.MP.Entities;
    using Senparc.Weixin.MP.Helpers;

    namespace Senparc.Weixin.MP.AdvancedAPIs
    {
        /// <summary>
        /// 客服接口
        /// </summary>
        public static class CustomerService
        {
            /// <summary>
            /// 微信公众平台客服接口地址
            /// </summary>
            private const string URL_FORMAT = "https://api.weixin.qq.com/cgi-bin/
message/custom/ send?access_token={0}";
            /// <summary>
            /// 发送文本信息
            /// </summary>
            /// <param name="accessToken"></param>
            /// <param name="openId">接收消息的用户 OpenId</param>
```

```
        /// <param name="content">要发送的文字信息</param>
        /// <returns></returns>
        public static WxJsonResult SendText(string accessToken, string
openId, string content)
        {
            var data = new
            {
                touser = openId,
                msgtype = "text",
                text = new
                {
                    content = content
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
        }
        /// <summary>
        /// 发送图片消息
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">接收消息的用户 OpenID</param>
        /// <param name="mediaId">已上传到微信的图片文件 ID</param>
        /// <returns></returns>
        public static WxJsonResult SendImage(string accessToken, string
openId, string mediaId)
        {
            var data = new
            {
                touser = openId,
                msgtype = "image",
                image = new
                {
                    media_id = mediaId
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
        }
        /// <summary>
        /// 发送语音消息
        /// </summary>
        /// <param name="accessToken"></param>
```

```
        /// <param name="openId">接收消息的用户 OpenID</param>
        /// <param name="mediaId">已上传到微信的语音文件 ID</param>
        /// <returns></returns>
        public static WxJsonResult SendVoice(string accessToken, string
openId, string mediaId)
        {
            var data = new
            {
                touser = openId,
                msgtype = "voice",
                voice = new
                {
                    media_id = mediaId
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
        }
        /// <summary>
        /// 发送视频消息
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">接收消息的用户 OpenID</param>
        /// <param name="mediaId">已上传到微信的视频文件 Id</param>
        /// <param name="title">视频消息的标题（非必须）</param>
        /// <param name="description">视频消息的描述（非必须）</param>
        /// <returns></returns>
        public static WxJsonResult SendVideo(string accessToken, string
openId, string mediaId, string title, string description)
        {
            var data = new
            {
                touser = openId,
                msgtype = "video",
                video = new
                {
                    media_id = mediaId,
                    title = title,
                    description = description
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
```

```
        }
        /// <summary>
        /// 发送音乐消息
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">接收消息的用户 OpenId</param>
        /// <param name="title">音乐标题（非必须）</param>
        /// <param name="description">音乐描述（非必须）</param>
        /// <param name="musicUrl">音乐链接</param>
        /// <param name="hqMusicUrl">高品质音乐链接，Wi-Fi 环境优先使用该链接播
放音乐</param>
        /// <param name="thumbMediaId">缩略图的媒体 ID</param>
        /// <returns></returns>
        public static WxJsonResult SendMusic(string accessToken, string
openId, string title, string description,string musicUrl, string hqMusicUrl,
string thumbMediaId)
        {
            var data = new
            {
                touser = openId,
                msgtype = "music",
                music = new
                {
                    title = title,
                    description = description,
                    musicurl = musicUrl,
                    hqmusicurl = hqMusicUrl,
                    thumb_media_id = thumbMediaId
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
        }
        /// <summary>
        /// 发送图文消息
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">接收消息的用户 OpenID</param>
        /// <param name="articles">图文消息实体，图文消息条数限制在 10 条以内
</param>
        /// <returns></returns>
```

```
        public static WxJsonResult SendNews(string accessToken, string
openId, List<Article> articles)
        {
            var data = new
            {
                touser = openId,
                msgtype = "news",
                news = new
                {
                    articles = articles.Select(z => new
                    {
                        title = z.Title,//标题
                        description = z.Description,//描述
                        url = z.Url,//单击后跳转的链接
                        picurl = z.PicUrl//图文消息的图片链接，支持 JPG、PNG 格式，
较好的效果为大图 640×320，小图 80×80
                    }).ToList()
                }
            };
            return ApiHelper.Post(accessToken, URL_FORMAT, data);
        }
    }
}
```

5.7　生成带参数的二维码接口

为了满足微信公众号对不同渠道推广效果分析的需要，微信公众平台提供了生成带参数二维码的接口。

5.7.1　带参数二维码接口简介

使用生成带参数二维码接口可以获得多个带不同场景值的二维码，微信个人用户扫描后，微信公众号可以接收到事件推送。

生成带参数二维码接口可以生成 2 种类型的二维码，分别是临时二维码和永久二维码。临时二维码有过期时间，最大为 1800 秒，但不限制生成不同临时二维码的数量。临时二维码主要适用于账号绑定等场景。永久二维码无过期时间，但

最多只能生成 100 000 个永久二维码。永久二维码主要适用于关注来源统计等场景。

获取带参数的二维码的过程包括两步，首先创建二维码 ticket，然后凭借 ticket 到指定 URL 换取二维码。

1. 创建二维码 ticket 接口

接口地址：https://api.weixin.qq.com/cgi-bin/qrcode/create?access_token= TOKEN

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求：JSON 数据包，参数描述见表 5-23。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-24。

每次创建二维码 ticket 需要提供一个自行设定的参数 scene_id。

表 5-23

参　数	说　明
expire_seconds	该二维码有效时间，以秒为单位。 最大不超过 1800 秒
action_name	二维码类型，QR_SCENE 为临时，QR_LIMIT_SCENE 为永久
action_info	二维码详细信息
scene_id	场景值 ID，临时二维码时为 32 位非 0 整型，永久二维码时最大值为 100 000（目前参数只支持 1～100 000）

表 5-24

参　数	说　明
ticket	获取二维码 ticket，凭借此 ticket 可以在有效时间内换取二维码
expire_seconds	当二维码类型为临时二维码时，表示临时二维码的有效时间，以秒为单位。最大不超过 1800 秒

2. 通过 ticket 换取二维码接口

接口地址：https://mp.weixin.qq.com/cgi-bin/showqrcode?ticket=TICKET

HTTP 请求方式：POST（请使用 HTTPS 协议）

请求参数：通过调用创建二维码 ticket 接口获取 ticket。

返回结果：二维码图片。

通过 ticket 换取二维码接口，接口无须登录即可调用。

5.7.2　带参数二维码接口封装

根据接口描述，首先定义创建二维码 ticket 结果 JSON 实体 CreateQrCodeResult。

```
/// <summary>
/// 二维码创建返回结果
/// </summary>
public class CreateQrCodeResult : WxJsonResult
{
    /// <summary>
    /// 获取二维码 ticket，凭借此 ticket 可以在有效时间内换取二维码
    /// </summary>
    public string ticket { get; set; }
    /// <summary>
    /// 二维码的有效时间，以秒为单位。最大不超过 1800 秒
    /// </summary>
    public int expire_seconds { get; set; }
}
```

然后新建一个类 QrCode，调用 ApiHelper 中的 POST 与 Download 方法，来实现创建二维码 ticket 接口 Create，以及通过 ticket 换取二维码接口 ShowQrCode

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using Senparc.Weixin.MP.Helpers;

namespace Senparc.Weixin.MP.AdvancedAPIs
{
    /// <summary>
    /// 二维码接口
    /// </summary>
    public static class QrCode
    {
        /// <summary>
```

```
        /// 创建二维码
        /// </summary>
        /// <param name="expireSeconds">该二维码有效时间，以秒为单位。最大不超过
1800 秒。0 时为永久二维码</param>
        /// <param name="sceneId">场景值 ID，临时二维码时为 32 位整型，永久二维码
时最大值为 1000 秒</param>
        /// <returns></returns>
        public static CreateQrCodeResult Create(string accessToken, int
expireSeconds, int sceneId)
        {
            //微信公众平台创建二维码的接口地址
            var urlFormat = "https://api.weixin.qq.com/cgi-bin/qrcode/create?
access_token={0}";
            object data = null;
            if (expireSeconds > 0)
            {
                //临时二维码
                data = new
                {
                    expire_seconds = expireSeconds,
                    action_name = "QR_SCENE",
                    action_info = new
                    {
                        scene = new
                        {
                            scene_id = sceneId
                        }
                    }
                };
            }
            else
            {
                //永久二维码
                data = new
                {
                    action_name = "QR_LIMIT_SCENE",
                    action_info = new
                    {
                        scene = new
                        {
                            scene_id = sceneId
```

```
                    }
                }
            };
        }
        return ApiHelper.Post<CreateQrCodeResult>(accessToken, urlFormat,
data);
    }
    /// <summary>
    /// 获取二维码（不需要 AccessToken）
    /// 在错误情况下（如 ticket 非法）返回 HTTP 错误码 404
    /// </summary>
    /// <param name="ticket">调用 Create 获取到的 ticket</param>
    /// <param name="stream">获取到的二维码图片</param>
    public static void ShowQrCode(string ticket, Stream stream)
    {
        //微信公众平台通过 ticket 换取二维码的接口地址
        var urlFormat = "https://mp.weixin.qq.com/cgi-bin/showqrcode?
ticket={0}";
        ApiHelper.Download(urlFormat, stream, ticket);
    }
  }
}
```

5.8　网页授权接口

网页授权接口允许微信公众号的第三方网页获取微信个人用户的基本信息，包括昵称、头像、性别、城市、国家、注册时间等。利用微信个人用户的基本信息，可以实现体验优化、用户来源统计、账号绑定、用户身份鉴权等功能。

5.8.1　网页授权接口简介

获取用户基本信息接口是在用户和公众号产生消息交互时，才能根据用户 OpenId 获取用户的基本信息。而通过网页授权的方式获取用户基本信息，则无需消息交互，只要微信个人用户进入到微信公众号的网页，就会弹出请求用户授权的界面，用户授权后，就可以获得其基本信息，此过程甚至不需要用户已经关注公众号。

网页授权接口是通过 OAuth 2.0 来完成网页授权的。

在微信公众号请求用户网页授权之前，微信公众号需要先到公众平台网站的“我的服务”页中配置授权回调域名，这里填写的域名不要加 http://，如图 5-4 所示。

授权回调域名配置规范为全域名，比如需要网页授权的域名为：www.qq.com，配置以后此域名下面的页面 http://www.qq.com/music.html 、 http://www.qq.com/login.html 都可以进行 OAuth 2.0 鉴权。但 http://pay.qq.com、http://music.qq.com、http://qq.com 无法进行 OAuth 2.0 鉴权。

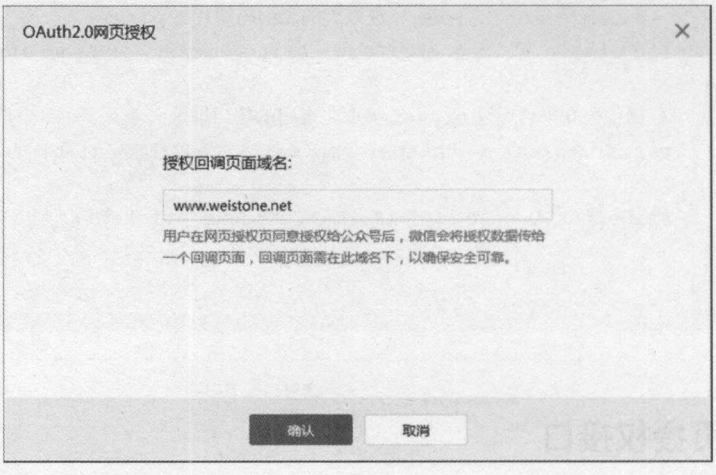


图 5-4

整个网页授权流程分为四步：

1. 引导用户进入授权页面同意授权，获取 code

在确保微信公众账号拥有授权作用域（scope 参数）的权限的前提下（服务号获得高级接口后，默认带有 scope 参数中的 snsapi_base 和 snsapi_userinfo），引导关注者打开如下页面：https://open.weixin.qq.com/connect/oauth2/authorize? appid=APPID&redirect_uri=REDIRECT_URI&response_type=code&scope=SCOPE&state=STATE#wechat_redirect

授权页面链接参数详细描述见表 5-25，用户打开链接后会进入如图 5-5 所示的授权页面。

如果用户同意授权，那么页面将跳转至 redirect_uri/?code=CODE&state=STATE。若用户禁止授权，则重定向后不会带上 code 参数，仅会带上 state 参数 redirect_uri?state=STATE。code 作为换取 access_token 的票据，每次用户授权带上的 code 将不一样，code 只能使用一次，5 分钟未被使用自动过期。

表 5-25　授权页面链接参数描述

参　数	是否必须	说　明
appid	是	公众号的唯一标识
redirect_uri	是	授权后重定向的回调链接地址，请使用 urlencode 对链接进行处理
response_type	是	返回类型，请填写 code
scope	是	应用授权作用域，snsapi_base （不弹出授权页面，直接跳转，只能获取用户 OpenId），snsapi_userinfo （弹出授权页面，可通过 OpenId 获得昵称、性别、所在地。并且，即使在未关注的情况下，只要用户授权，也能获取其信息）
state	否	重定向后会带上 state 参数，开发者可以填写 a-zA-Z0-9 的参数值
#wechat_redirect	是	无论直接打开还是做页面 302 重定向时候，必须带此参数

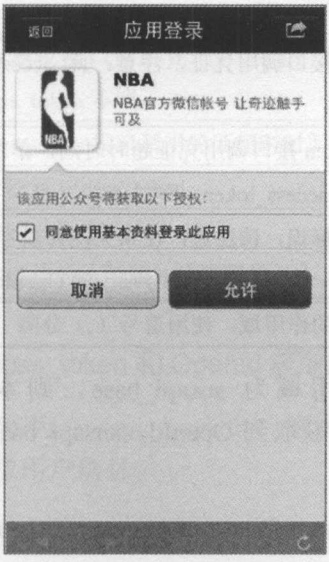


图 5-5

2. 通过 code 换取网页授权 access_token

这里通过 code 换取的网页授权 access_token，与基础支持中的 access_token 不同。微信公众号可通过下述接口来获取网页授权 access_token。

接口地址：https://api.weixin.qq.com/sns/oauth2/access_token?appid= APPID&secret=SECRET&code=CODE&grant_type=authorization_code

HTTP 请求方式：GET（请使用 HTTPS 协议）

请求参数描述：见表 5-26。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-27。

表 5-26

参　数	是否必须	说　明
appid	是	公众号的唯一标识
secret	是	公众号的 appsecret
code	是	填写第一步获取的 code 参数
grant_type	是	填写为 authorization_code

表 5-27

参　数	描　述
access_token	网页授权接口调用凭证，注意：此 access_token 与基础支持的 access_token 不同
expires_in	access_token 接口调用凭证超时时间，单位（秒）
refresh_token	用户刷新 access_token
openid	用户唯一标识，请注意，在未关注公众号时，用户访问公众号的网页，也会产生一个用户和公众号唯一的 OpenId
scope	用户授权的作用域，使用逗号（,）分隔

如果网页授权的作用域为 snsapi_base，则本步骤中获取到网页授权 access_token 的同时，也会获取到 OpenId，snsapi_base 式的网页授权流程即到此为止。

3. 如果需要，则可以刷新网页授权 access_token，避免过期

由于 access_token 拥有较短的有效期，当 access_token 超时后，可以使用 refresh_token 进行刷新，refresh_token 拥有较长的有效期（7 天、30 天、60 天、90 天），当 refresh_token 失效之后，需要用户重新授权。

接口地址：https://api.weixin.qq.com/sns/oauth2/refresh_token?appid= APPID&grant_type=refresh_token&refresh_token=REFRESH_TOKEN

HTTP 请求方式：GET（请使用 HTTPS 协议）

请求参数描述：见表 5-28。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-29。

表 5-28

参　数	是否必须	说　明
appid	是	公众号的唯一标识
grant_type	是	填写为 refresh_token
refresh_token	是	填写通过 access_token 获取到的 refresh_ token 参数

表 5-29

参　数	描　述
access_token	网页授权接口调用凭证，注意：此 access_token 与基础支持的 access_token 不同
expires_in	access_token 接口调用凭证超时时间，单位（秒）
refresh_token	用户刷新 access_token
openid	用户唯一标识
scope	用户授权的作用域，使用逗号（,）分隔

4. 通过网页授权 access_token 和 OpenId 获取用户基本信息

如果网页授权作用域为 snsapi_userinfo，则此时微信公众号可以通过 access_token 和 OpenId 获取用户信息。

接口地址：https://api.weixin.qq.com/sns/userinfo?access_token= ACCESS_ TOKEN&openid=OPENID&lang=zh_CN

HTTP 请求方式：GET（请使用 HTTPS 协议）

请求参数描述：见表 5-30。

返回结果：在正确情况下将返回 JSON 数据包，返回结果参数描述见表 5-31。

表 5-30

参　数	描　述
access_token	网页授权接口调用凭证，注意：此 access_token 与基础支持的 access_token 不同
openid	用户的唯一标识
lang	返回国家地区语言版本，zh_CN 简体、zh_TW 繁体、en 英语

表 5-31

参　数	描　述
openid	用户的唯一标识
nickname	用户昵称
sex	用户的性别，值为 1 时是男性，值为 2 时是女性，值为 0 时是未知
province	用户个人资料填写的省份
city	普通用户个人资料填写的城市
country	国家，如中国为 CN
headimgurl	用户头像，最后一个数值代表正方形头像大小（有 0、46、64、96、132 数值可选，0 代表 640×640 正方形头像），用户没有头像时该项为空
privilege	用户特权信息，JSON 数组，如微信沃卡用户为（chinaunicom）

5.8.2 网页授权接口封装

根据接口描述，首先定义网页授权 AccessToken 的 JSON 数据实体 OAuthAccessTokenResult，以及通过网页授权获取的用户基本信息的 JSON 数据实体 OAuthUserInfo。

```
/// <summary>
/// 获取 OAuth 网页授权 AccessToken 的结果
/// 如果错误，则返回结果{"errcode":40029,"errmsg":"invalid code"}
/// </summary>
public class OAuthAccessTokenResult : WxJsonResult
```

```
    {
        /// <summary>
        /// 网页授权接口调用凭证，注意：此 access_token 与其他接口的 access_token 不同
        /// </summary>
        public string access_token { get; set; }
        /// <summary>
        /// 接口调用凭证超时时间，单位（秒）
        /// </summary>
        public int expires_in { get; set; }
        /// <summary>
        /// 用户刷新 access_token
        /// </summary>
        public string refresh_token { get; set; }
        /// <summary>
        /// 用户唯一标识，请注意，在未关注公众号时，用户访问公众号的网页，也会产生一个用户和公众号唯一的 OpenId
        /// </summary>
        public string openid { get; set; }
        /// <summary>
        /// 应用授权作用域，snsapi_base（不弹出授权页面，直接跳转，只能获取用户 OpenId），snsapi_userinfo （弹出授权页面，可通过 OpenId 获得昵称、性别、所在地。并且，即使在未关注的情况下，只要用户授权，也能获取其信息）
        /// </summary>
        public string scope { get; set; }
    }
    /// <summary>
    /// 通过 OAuth 获取到的用户信息（snsapi_userinfo=scope）
    /// </summary>
    public class OAuthUserInfo
    {
        /// <summary>
        /// 用户的唯一标识
        /// </summary>
        public string openid { get; set; }
        /// <summary>
        /// 用户昵称
        /// </summary>
        public string nickname { get; set; }
        /// <summary>
```

```
        /// 用户的性别，值为1时是男性，值为2时是女性，值为0时是未知
        /// </summary>
        public int sex { get; set; }
        /// <summary>
        /// 用户所在省份
        /// </summary>
        public string province { get; set; }
        /// <summary>
        /// 用户所在城市
        /// </summary>
        public string city { get; set; }
        /// <summary>
        /// 用户所在国家
        /// </summary>
        public string country { get; set; }
        /// <summary>
        /// 用户头像，最后一个数值代表正方形头像大小（有0、46、64、96、132数值可选，0代表640×640正方形头像），用户没有头像时该项为空
        /// </summary>
        public string headimgurl { get; set; }
        /// <summary>
        /// 用户特权信息，JSON 数组，如微信沃卡用户为（chinaunicom）
        /// 作者注：其实这个格式称不上JSON，只是一个单纯数组
        /// </summary>
        public string[] privilege { get; set; }
    }
```

然后新建一个类OAuth，调用RequestUtility的HttpGet方法进行HTTP请求，并使用ApiHelper类的GetResult方法对返回结果进行反序列化，实现网页授权流程的四个步骤。

```
using Senparc.Weixin.MP.Entities;
using Senparc.Weixin.MP.Helpers;
using Senparc.Weixin.MP.HttpUtility;

namespace Senparc.Weixin.MP.AdvancedAPIs
{
    /// <summary>
    /// 应用授权作用域
    /// </summary>
    public enum OAuthScope
    {
```

```
            /// <summary>
            /// 不弹出授权页面，直接跳转，只能获取用户 OpenId
            /// </summary>
            snsapi_base,
            /// <summary>
            /// 弹出授权页面，可通过 OpenId 获得昵称、性别、所在地
            /// 并且，即使在未关注的情况下，只要用户授权，也能获取其信息
            /// </summary>
            snsapi_userinfo
        }
        /// <summary>
        /// OAuth 2.0 网页授权接口
        /// </summary>
        public static class OAuth
        {
            /// <summary>
            /// 第一步：生成网页授权访问地址
            /// </summary>
            /// <param name="appId">公众号的唯一标识</param>
            /// <param name="redirectUrl">授权后重定向的回调链接地址，请使用
urlencode 对链接进行处理</param>
            /// <param name="state">重定向后会带上 state 参数，可以填写 a-zA-Z0-9 的
参数值</param>
            /// <param name="scope">应用授权作用域</param>
            /// <param name="responseType">返回类型，目前只有 code 一种</param>
            /// <returns>网页授权访问地址，引导关注者打开该地址进入网页授权</returns>
            public static string GetAuthorizeUrl(string appId, string redirectUrl,
string state, OAuthScope scope, string responseType = "code")
            {
                var                                                    url
=string.Format("https://open.weixin.qq.com/connect/oauth2/authorize?appid={
0}&redirect_uri={1}&response_type={2}&scope={3}&state={4}#wechat_redirect",
appId, System.Web.HttpUtility.UrlEncode(redirectUrl), responseType, scope,
state);
                return url;
            }
            /// <summary>
            /// 第二步：获取 AccessToken
            /// 用户访问第一步网页授权页面后，无论同意或拒绝，都会返回 redirectUrl 页面
            /// 如果用户同意授权，页面将跳转至 redirect_uri/?code=CODE&state=STATE
```

```
        /// 若用户禁止授权，则重定向后不会带上 code 参数，仅会带上 state 参数
redirect_uri?state=STATE
        /// </summary>
        /// <param name="appId">公众号的 appId</param>
        /// <param name="secret">公众号的 appsecret</param>
        /// <param name="code">code 作为换取 access_token 的票据，每次用户授权带
上的 code 将不一样
        /// code 只能使用一次，5 分钟未被使用自动过期。</param>
        /// <param name="grantType">目前只有 authorization_code 一种</param>
        /// <returns></returns>
        public static OAuthAccessTokenResult GetAccessToken(string appId,
string secret, string code, string grantType = "authorization_code")
        {
            var url =string.Format("https://api.weixin.qq.com/sns/oauth2/
access_token?appid={0}&secret={1}&code={2}&grant_type={3}",appId, secret, code,
grantType);
            string result = RequestUtility.HttpGet(url, null);
            return ApiHelper.GetResult<OAuthAccessTokenResult>(result);
        }
        /// <summary>
        /// 第三步：刷新 access_token（如果需要）
        /// 由于 access_token 拥有较短的有效期，当 access_token 超时后，可以使用
refresh_token 进行刷新
        /// refresh_token 拥有较长的有效期(7 天、30 天、60 天、90 天)，当 refresh_token
失效后，需要用户重新授权
        /// </summary>
        /// <param name="appId">公众号的 appId</param>
        /// <param name="refreshToken">填写通过第二步获取到的 refresh_token 参
数</param>
        /// <param name="grantType"></param>
        /// <returns></returns>
        public static OAuthAccessTokenResult RefreshToken(string appId,
string refreshToken, string grantType = "refresh_token")
        {
            var                                                          url
=string.Format("https://api.weixin.qq.com/sns/oauth2/refresh_token?appid={0
}&grant_type={1}&refresh_token={2}",appId, grantType, refreshToken);
            string result = RequestUtility.HttpGet(url, null);
            return ApiHelper.GetResult<OAuthAccessTokenResult>(result);
        }
        /// <summary>
```

```
        /// 第四步：拉取用户信息（需 scope 为 snsapi_userinfo）
        /// </summary>
        /// <param name="accessToken"></param>
        /// <param name="openId">要获取使用信息的用户 OpenId</param>
        /// <returns></returns>
        public static OAuthUserInfo GetUserInfo(string accessToken,string
openId)
        {
            var url =string.Format("https://api.weixin.qq.com/sns/userinfo?
access_token={0}&openid={1}",accessToken,openId);
            string result = RequestUtility.HttpGet(url, null);
            return ApiHelper.GetResult<OAuthUserInfo>(result);
        }
    }
}
```

第6章
商用案例1——预约系统

本章主要讲解以规范的商用系统架构，在Microsoft Visual Studio 2012开发环境中，使用C#语言开发一套适应微信公众号各种商用环境的预约系统。

6.1 预约系统需求

在企业实际使用与运营微信公众号的过程中，为了更好地通过微信公众号服务客户，根据企业自身业务特点，会有各种各样需要微信个人用户填写预约、预定、申请、报名等表单的需求。

对于一个医院，会有使用微信公众号进行挂号预约的功能需求，以方便病人在线挂号。对于一个汽车4S店，会有使用微信公众号进行试驾预约的功能需求，以方便意向客户在线提交试驾申请。对于一个培训机构，会有使用微信公众号进行报名的功能需求，以方便学员在线报名。对于一个房产公司或房产中介机构，

会有使用微信公众号进行预约看房的功能需求，以方便客户在线进行看房预约。

每个企业都会有使用微信公众号提供预约服务的需求，而预约服务本质上是请微信个人用户填写预约表单。每个企业业务各不相同，对一个企业来说，对不同客户提供的服务也不一样，不同发展时期的业务范围也不一样。不同的业务，需要设计供用户填写的预约表单都是不一样的。

如果对每个业务，都去开发一套仅针对该业务的预约系统，那么开发成本是很高的。

设计并实现一套能适用于各种不同业务的通用预约系统，对运营微信公众号的企业来说，可以节省开发成本。同时，使用通用预约系统，进行简单配置，即可生成面向不同业务的预约表单，这将大大提高微信公众号的运营效率。

6.2　预约系统功能及设计

6.2.1　预约系统功能

预约系统主要由以下几部分构成：

1. 微信个人用户在微信公众号中使用的预约前台

（1）供微信个人用户填写的预约表单。

（2）微信个人用户查看已填写的预约表单列表。

（3）微信个人用户修改已填写的预约表单。

2. 微信公众号使用的预约管理后台

（1）微信公众号对预约表单进行设计，生成新的预约。

（2）微信公众号对已设计预约表单进行修改、删除、设置过期等管理。

（3）微信公众号查看某个预约已填写的预约申请及填写预约的客户信息。

（4）微信公众号对已填写的预约进行审核等管理。

微信个人用户通过单击自定义菜单或发送消息，进入预约功能，进行预约表单填写。预约表单需要填写的内容与预约填写说明，由微信公众号在后台设置。

限于篇幅所限，本章仅给出一个通用预约系统的示例程序实现，包含通用预约系统的核心功能。读者参照本章的示例程序，可以非常方便地搭建出正式商用的通用预约系统。

6.2.2 不定字段数目的数据库表和数据结构设计

一个商用系统的实现步骤一般为确定需求、根据需求进行数据库设计、根据数据库结构进行程序设计。如果需求改变，则需要重新进行数据库设计，然后根据新的数据库结构修改程序。

一般的商用系统数据库都是关系型数据库。在数据库设计时，需要确定整个系统有哪些表，每个表有哪些字段。限于关系型数据库的结构限制，任何表的字段改变，都需要重新修改数据库，并且修改程序的业务逻辑。

因此，通常我们实现一个商用系统，一定是先确定好需求再实现整个系统。而对通用预约系统来说，需求是变化的，每种业务类型所需要的预约表单各不相同，在实现系统前，是没有办法确定预约表单的需求的，也就是说，没有办法在数据库设计时，确定预约表单的数据结构。

这种在开发中无法确定数据库结构的情况，我们称为“不定字段数目的数据库表和数据结构设计”问题。

不定字段数目的数据库表和数据结构一般有两种解决方案。

1. 预留足够的空白字段

预留空白字段的基本原理就是在数据库表设计的时候加入一些多余的字段。数据库定义代码如下：

```
CREATE TABLE Sample(
    name varchar(12),
    --扩展字段 0
    field0 varchar(1),
    --扩展字段 1
    field1 varchar(1),
```

```
    --扩展字段 N
    fieldN varchar(1)
}
```

然后看实际运行时的需要，动态分配字段给系统使用。在程序中需要一个这样的结构来描述分配情况：

```
public class Available
{
    //未使用的扩展字段个数
    public int CurrentUnusedFieldNumber;
    //扩展字段的真实名称
    public Hashtable FieldToRealName;
}
```

在某一时刻，数据状况可能是这样的：

CurrentUnusedFieldNumber=3

哈西表 FieldToRealName 包含内容是("field0"="SomeId", "field1"="AnyName", "field2=IsOk")

2. 改列为行，用另外一个表存放定制字段

这种方式的基本原理是使用两个表来表示一个不定长度的表，一个表保存固定信息，将不能确定个数和具体含义的字段放到另一个表中。数据库定义代码如下：

```
CREATE TABLE SampleFields
(
    id Integer,
    name varchar(30)
)
CREATE TABLE SampleFields
(
    idSample Integer,
    fieldName varchar(30),
    fieldValue varchar(100)
)
```

其中 idSample 是关联到 Sample 表的 ID 字段。

两种方式各有优缺点。第一种方式符合关系型数据库的结构特点，查询效率较高，但当实际需要的字段超过预定义的空白字段个数时，该方案将无法适应需求。而第二种方案没有字段个数的限制，但在数据量非常大时，查询效率会比较低。

根据通用预约表单的特点，我们无法预先确定预约表单的字段个数。如果采用第一种方案，那么我们可以预先定义 10 个空白字段，这对一般的预约可能是足够的。但是，如果我们想使用通用预约系统来实现一个提交简历的功能，10 个空白字段就远远不能满足需求了。

本章的通用预约系统示例程序，将采用第二种方式实现。

6.2.3 数据表设计

根据预约业务的特点，我们需要设计 2 张表，一张表用来存放预约表单需要填写的那些信息，即预约表单的结构；另一张表用来存放用户填写的预约申请信息。

我们采用“改列为行，用另外一个表存放定制字段”的解决方案来实现通用预约表单的设计。最后的数据表为 4 张表：

1. 预约表单表 Reservation。
2. 预约表单具体包含的字段表 ReservationContent。
3. 用户填写的预约表单内容表 UserReservation。
4. 用户填写的预约表单中每个字段具体填写的内容表 UserReservationContent。

生成数据库的 SQL 语句如下：

```
-- ------------------------------------------------
-- Creating all tables
-- ------------------------------------------------
--预约表单表
-- Creating table 'Reservation'
CREATE TABLE [Reservation] (
    --主键
   [ID] int IDENTITY(1,1) NOT NULL,
    --预约表单名称
```

```
    [Name] nvarchar(4000)  NOT NULL
);
GO
--预约表单具体包含的字段表
-- Creating table 'ReservationContent'
CREATE TABLE [ReservationContent] (
    --主键
    [ID] int IDENTITY(1,1) NOT NULL,
    --字段名称
    [Name] nvarchar(4000)  NOT NULL,
    --字段初始内容
    [Content] nvarchar(4000)  NOT NULL,
    --预约表单 ID
    [ReservationID] int  NOT NULL
);
GO
--用户填写的预约表单内容表
-- Creating table 'UserReservation'
CREATE TABLE [UserReservation] (
    --主键
    [ID] int IDENTITY(1,1) NOT NULL,
    --填写表单的微信个人用户的 OpenId
    [WeixinOpenId] nvarchar(4000)  NOT NULL,
    --预约表单 ID
    [ReservationID] int  NOT NULL
);
GO
--用户填写的预约表单中每个字段具体填写的内容表
-- Creating table 'UserReservationContent'
CREATE TABLE [UserReservationContent] (
    --主键
    [ID] int IDENTITY(1,1) NOT NULL,
    --字段具体填写内容
    [Content] nvarchar(4000)  NOT NULL,
    --用户填写的预约表单 ID
    [UserReservationId] int  NOT NULL,
    --预约表单具体包含的字段 ID
    [ReservationContentId] int  NOT NULL
);
GO
-- --------------------------------------------------
```

```
-- Creating all PRIMARY KEY constraints
-- --------------------------------------------------
-- Creating primary key on [ID] in table 'ReservationContent'
ALTER TABLE [ReservationContent]
ADD CONSTRAINT [PK_ReservationContent]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'UserReservationContent'
ALTER TABLE [UserReservationContent]
ADD CONSTRAINT [PK_UserReservationContent]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'UserReservation'
ALTER TABLE [UserReservation]
ADD CONSTRAINT [PK_UserReservation]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'Reservation'
ALTER TABLE [Reservation]
ADD CONSTRAINT [PK_Reservation]
   PRIMARY KEY ([ID] );
GO
```

6.3 预约系统架构实现

6.3.1 商用系统三层架构简述

在商用系统开发中，三层架构是最基本的，也是使用最广泛的系统架构。

三层架构就是将整个业务应用划分为：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。区分层次的目的即为了实现“高内聚，低耦合”的思想，如图 6-1 所示。

一个完整的三层架构包括：

1. 数据访问框架层（ORM）

该层主要实现 DAL 层与数据库交互的数据传输。

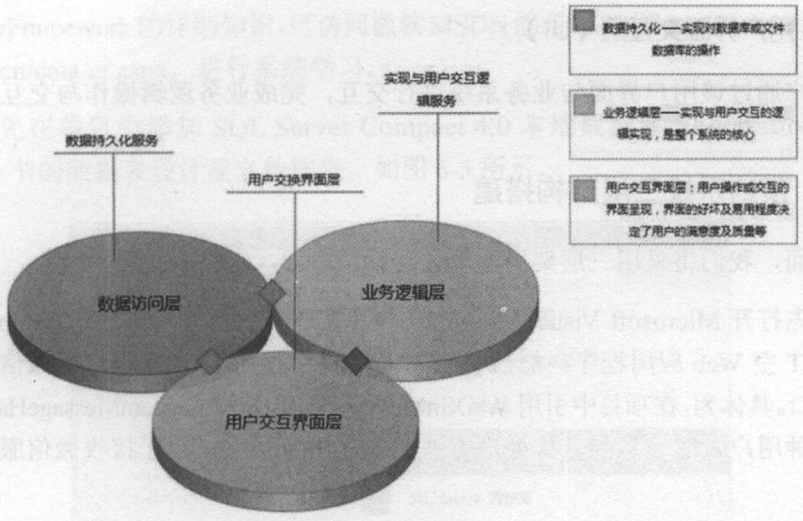


图 6-1

2. 数据实体定义层（DataEntities）

该层主要定义与数据库一致的数据实体，用于 ORM 通过该实体实现与数据库交互的数据传输。在 EntityFramework、Nhibernate 等完善的 ORM 框架中，已包含该层。

3. 数据访问层（DAL）

该层通过调用 ORM，实现数据持久化，可以支持多种数据库，如 sqlserver、oracle、mysql、sqlite 等。

4. 视图实体定义层（ViewEntities）

该层主要定义用于用户界面交互层的数据实体，以及数据实体与视图实体的转换方法，实现数据库与用户界面交互的隔离。

5. 业务逻辑层（BLL）

该层通过调用视图实体层、数据访问层，实现整个业务系统的核心功能，完成系统业务的处理。

6. 用户界面交互层（UI）

用户通过该用户界面与业务系统进行交互，完成业务逻辑操作与交互。

6.3.2 预约系统三层架构搭建

下面，我们将采用三层架构实现预约系统。

首先打开 Microsoft Visual Studio 2012 开发环境，建立一个名为 Sample_3 的 ASP.NET 空 Web 应用程序。然后参照第 4 章的 Sample_2 项目建立好微信公众号访问接口。具体为，在项目中引用 WeiXinMessageSDK，建立 CustomMessageHandler.cs 重写每种用户发送消息的具体处理方法，建立 Index.aspx 用于接收微信服务器请求。

接下来在 Sample_3 项目中建立用于存放各层代码的文件夹 ORM、DAL、ViewEntities、BLL 等。

Sample_3 项目的项目结构如图 6-2 所示。

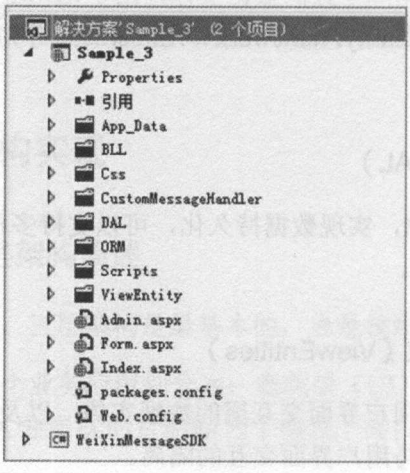


图 6-2

6.3.3 实现数据访问框架

数据访问框架层（ORM）采用.NET 平台的 EntityFramework 框架来实现。关

于 EntityFramework 的详细知识，可访问微软 MSDN 的帮助页面 http://msdn.microsoft.com/zh-cn/data/ef.aspx，进行系统学习。

首先在项目中添加 SQL Server Compact 4.0 本地数据库 Reservation.sdf。按 6.2.3 节的数据表设计建立数据表，如图 6-3 所示。

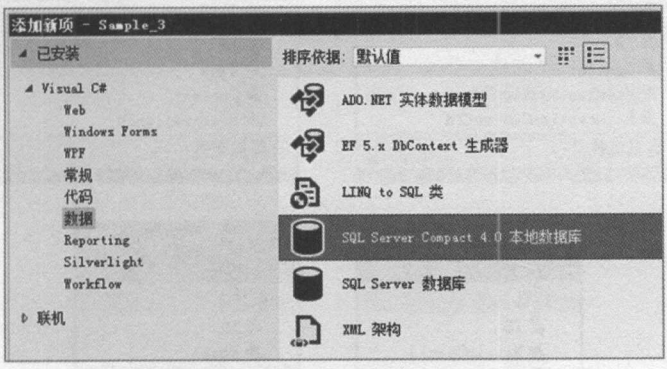


图 6-3

然后在 ORM 文件夹中建立 ADO.NET 实体数据模型 Reservation.edmx。双击 Reservation.edmx 打开实体数据模型设计器，单击鼠标右键选择“从数据更新模型”，选择“从数据库生成”。单击“下一步”按钮后，新建一个“Microsoft SQL Server Compact 4.0”连接，指向前面建好的数据库 Reservation.sdf。

选中数据库的所有表，生成实体数据模型，如图 6-4 所示。

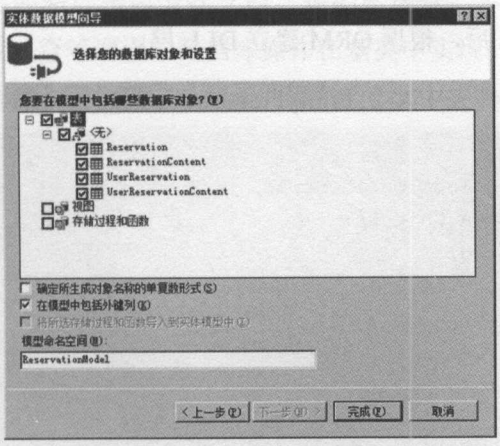


图 6-4

生成完毕后，EntityFramwork 已自动为我们创建好 ORM 及 DataEntities，如图 6-5 所示。



图 6-5

6.3.4 实现数据访问层

在建立好 ORM 后，根据 ORM 建立 DLL 层。

首先在 DLL 文件夹中建立 EntityFramework 的连接方法 DbContextFactory。

```
using System;
using System.Collections.Generic;
using System.Data.Objects;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Web;
using Sample_3.ORM;
namespace Sample_3
{
    class DbContextFactory
    {
```

```
        /// <summary>
        /// 实现对 EF 上下文实例进行管理，保证线程内唯一
        /// </summary>
        /// <returns></returns>
        public static ObjectContext GetCurrentDbContext()
        {
            //CallContext 是类似于方法调用的线程本地存储区的专用集合对象，并提供对每
个逻辑执行线程都唯一的数据槽。数据槽不在其他逻辑线程上的调用上下文之间共享。当 CallContext
沿执行代码路径往返传播并且由该路径中的各个对象检查时，可将对象添加到其中
            //当对另一个 AppDomain 中的对象进行远程方法的调用时，CallContext 类将
生成一个与该远程调用一起传播的 LogicalCallContext 实例。只有公开 ILogicalThread
Affinative 接口并存储在 CallContext 中的对象被在 LogicalCallContext 中传播到
AppDomain 外部。不支持此接口的对象不在 LogicalCallContext 实例中与远程方法调用一起传输
            //暂时未考虑跨域情况
            ObjectContext context = (ObjectContext)CallContext.GetData
("ReservationContent");
            if (context == null)
            {
                context = new ReservationContainer();
                context.Connection.Open();
                CallContext.SetData("ReservationContent", context);
            }
            return context;
        }
    }
}
```

在建立各数据表的数据访问方法之前，我们先建立一个抽象数据访问基类 BaseDll，增、删、改、查等通用的数据库操作在基类中实现。具体数据表的数据访问类只需要继承该基类并实现自己特定的数据访问方法即可。

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.Objects;
using System.Data.Objects.DataClasses;
using System.Linq;
using System.Linq.Expressions;
using Sample_3.ORM;
namespace Sample_3
```

```
    {
        /// <summary>
        /// 仓储基类，用于数据访问
        /// </summary>
        /// <typeparam name="T"></typeparam>
        public abstract class BaseDll<T> where T : EntityObject
        {
            //EF上下文
            public virtual ObjectContext DbContext
            {
                get { return DbContextFactory.GetCurrentDbContext(); }
            }
            #region 新增实体
            /// <summary>
            /// 新增实体
            /// </summary>
            /// <param name="entity">实体</param>
            /// <param name="isSaveChage"> 是否保存改变</param>
            /// <returns></returns>
            public T AddEntity(T entity, bool isSaveChage = true)
            {
                if (DbContext.IsAttached(entity))
                {
                    DbContext.CreateObjectSet<T>().Detach(entity);
                }
                DbContext.CreateObjectSet<T>().Attach(entity);
                DbContext.ObjectStateManager.ChangeObjectState(entity,
EntityState.Added);
                if (isSaveChage)
                {
                    DbContext.SaveChanges();
                }
                return entity;
            }
            #endregion
            #region 修改实体
            /// <summary>
            /// 修改实体
            /// </summary>
            /// <param name="entity">实体</param>
            /// <param name="isSaveChage">是否保存改变 </param>
```

```
        /// <returns></returns>
        public bool UpdateEntity(T entity, bool isSaveChage = true)
        {
            if (DbContext.IsAttached(entity))
            {
                DbContext.CreateObjectSet<T>().Detach(entity);
            }
            DbContext.CreateObjectSet<T>().Attach(entity);
            DbContext.ObjectStateManager.ChangeObjectState(entity,
EntityState.Modified);
            if (isSaveChage)
            {
                return DbContext.SaveChanges() > 0;
            }
            return true;
        }
        #endregion
        #region 删除实体
        /// <summary>
        /// 删除实体
        /// </summary>
        /// <param name="entity">实体</param>
        /// <param name="isSaveChange">是否保存改变 </param>
        /// <returns></returns>
        public bool DeleteEntity(T entity, bool isSaveChange = true)
        {
            if (DbContext.IsAttached(entity))
            {
                DbContext.CreateObjectSet<T>().Detach(entity);
            }
            DbContext.CreateObjectSet<T>().Attach(entity);
            DbContext.ObjectStateManager.ChangeObjectState(entity,
EntityState.Deleted);
            if (isSaveChange)
            {
                return DbContext.SaveChanges() > 0;
            }
            return true;
        }
        #endregion
        #region 保存改变
```

```
        /// <summary>
        /// 保存改变
        /// </summary>
        /// <returns></returns>
        public int SaveChanges()
        {
            return DbContext.SaveChanges();
        }
        #endregion
        #region 查询单个实体
        /// <summary>
        /// 查询单个实体
        /// </summary>
        /// <param name="expression"></param>
        /// <returns></returns>
        public T LoadEntity(Expression<Func<T, bool>> expression)
        {
            IQueryable<T> entitys = LoadEntities(expression);
            T entity = entitys.FirstOrDefault();
            if (entity != null)
            {
                if (DbContext.IsAttached(entity))
                    DbContext.Detach(entity);
            }
            return entity;
        }
        #endregion
        #region 查询实体
        /// <summary>
        /// 查询实体
        /// </summary>
        /// <param name="expression"></param>
        /// <returns></returns>
      public IQueryable<T> LoadEntities(Expression<Func<T, bool>> expression)
        {
            return
DbContext.CreateObjectSet<T>().Where<T>(expression).AsQueryable();
        }
        #endregion
    }
    public static class EntityObjectExt
```

```
        {
            /// <summary>
            /// 判断传入的对象是否已附加到当前实体数据对象中
            /// </summary>
            /// <param name="context">实体数据对象</param>
            /// <param name="entity">对象</param>
            /// <returns>是否附加成功</returns>
            public static bool IsAttached(this ObjectContext context, object
entity)
            {
                if (entity == null)
                {
                    throw new ArgumentNullException("entity is null");
                }
                try
                {
                    var dd = entity as EntityObject;
                    ObjectStateEntry entry = context.ObjectStateManager.
GetObjectStateEntry(dd.EntityKey);
                    return (entry.State != EntityState.Detached);
                }
                catch
                {
                    return false;
                }
            }
        }
    }
```

在建立好数据访问基类后，通过继承 BaseDll，实现预约表单表的数据访问类 ReservationDll、预约表单具体包含的字段表的数据访问类 ReservationContentDll、用户填写的预约表单内容表的数据访问类 UserReservationDll、用户填写的预约表单中每个字段具体填写内容表的数据访问类 UserReservationContent。

该演示程序仅支持对同一个预约表单进行修改，通过 ReservationDll 类的 AddReservation 方法及 ReservationContentDll 类的 AddReservationContent 方法，确保所有提交的预约表单仅存放在数据库 Reservation 表的第一条记录中。新提交的预约表单将覆盖旧的预约表单。

在实际项目中，可通过重写上述两个方法，实现对多个预约表单的支持。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Sample_3.ORM;
namespace Sample_3
{
    //预约表单表的数据访问类
    public class ReservationDll : BaseDll<Reservation>
    {
        /// <summary>
        /// 本演示程序仅保存一个预约表单
        /// </summary>
        const int ID = 1;
        /// <summary>
        /// 添加预约表单
        /// </summary>
        /// <param name="entity">预约表单</param>
        /// <returns>返回 0 表示添加失败</returns>
        public int AddReservation(Reservation entity)
        {
            var temp = base.LoadEntity(p => p.ID == ID);
            if (temp != null)
            {
                temp.Name = entity.Name;
                if (base.UpdateEntity(temp))
                    return ID;
                else
                    return 0;
            }
            else
            {
                return base.AddEntity(entity).ID;
            }
        }
    }
    //预约表单具体包含的字段表的数据访问类
    public class ReservationContentDll : BaseDll<ReservationContent>
    {
        /// <summary>
        /// 添加预约表单字段
```

```
        /// </summary>
        /// <param name="entities"></param>
        /// <returns>返回 false 表示添加失败</returns>
        public bool AddReservationContent(List<ReservationContent> entities)
        {
            if (entities == null || entities.Count == 0)
                throw new ArgumentNullException("entities 不能为空");
            int reservationID = entities.First().ReservationID;
            //获取原有预约表单字段
            var oldEntities = base.LoadEntities(p => p.ReservationID ==
reservationID).ToList();
            foreach (var entity in oldEntities)
            {
                //删除任一原有预约表单字段失败，则操作失败
                if (!base.DeleteEntity(entity))
                    return false;
            }
            foreach (var entity in entities)
            {
                //添加任一新预约表单字段失败，则操作失败
                if (base.AddEntity(entity).ID <= 0)
                    return false;
            }
            return true;
        }
    }
    //用户填写的预约表单内容表的数据访问类
    public class UserReservationContentDll : BaseDll<UserReservationContent>
    {
        /// <summary>
        /// 添加用户填写的预约表单字段内容
        /// </summary>
        /// <param name="entities"></param>
        /// <returns>返回 false 表示添加失败</returns>
        public bool AddUserReservationContent(List<UserReservationContent>
entities)
        {
            if (entities == null || entities.Count == 0)
                throw new ArgumentNullException("entities 不能为空");
            foreach (var entity in entities)
            {
```

```
                //添加任一用户填写的预约表单字段内容失败，则操作失败
                if (base.AddEntity(entity).ID <= 0)
                    return false;
            }
            return true;
        }
    }
     //用户填写的预约表单中每个字段具体填写内容表的数据访问类
    public class UserReservationDll : BaseDll<UserReservation>
    {
    }
}
```

6.3.5 实现视图实体层

视图实体层主要定义供业务逻辑层和用户界面交互层调用的实体。同时，需要定义数据实体与视图实体的转换方法。

首先，定义一个接口 IViewModel，来规定数据实体与视图实体的转换方法。

```
using System;
using System.Collections.Generic;
using System.Data.Objects.DataClasses;
using System.Linq;
using System.Web;
namespace Sample_3
{
    /// <summary>
    /// 数据实体与视图实体的转换
    /// </summary>
    /// <typeparam name="TV">数据实体对应的视图实体类型</typeparam>
    /// <typeparam name="TD">数据实体类型</typeparam>
    public interface IViewModel<TV, TD> where TD : EntityObject
    {
        /// <summary>
        /// 数据实体转换为视图实体
        /// </summary>
        /// <param name="entity">数据实体</param>
        /// <returns>视图实体</returns>
        TV GetViewModel(TD entity);
        /// <summary>
```

```
        /// 视图实体转换为数据实体
        /// </summary>
        /// <param name="entity">视图实体</param>
        /// <returns>数据实体</returns>
        TD GetDataEntity(TV entity);
    }
}
```

然后，定义每个数据实体对应的视图实体，并在视图实体中实现转换接口 IViewModel。实体数据的转换使用开源的 Object-Object Mapping 工具 AutoMapper。AutoMapper 是一个.NET 的对象映射工具，主要用于领域对象与 DTO 之间的转换、数据库查询结果映射至实体对象。关于 AutoMapper 的详细介绍，可在 AutoMapper 项目的官方 Wiki 中获取：https://github.com/AutoMapper/AutoMapper/wiki。

```
using System;
using System.ComponentModel.DataAnnotations;
using Sample_3.ORM;
namespace Sample_3
{
    /// <summary>
    /// 预约表单
    /// </summary>
    [Serializable]
    public class ReservationEntity : IViewModel<ReservationEntity, Reservation>
    {
        static ReservationEntity()
        {
            AutoMapper.Mapper.CreateMap<Reservation, ReservationEntity>();
            AutoMapper.Mapper.CreateMap<ReservationEntity, Reservation>();
        }
        /// <summary>
        /// ID
        /// </summary>
        [Key]
        public int ID { get; set; }
        /// <summary>
        /// 预约表单名称
        /// </summary>
        public string Name { get; set; }
        public ReservationEntity GetViewModel(Reservation entity)
        {
```

```
            return AutoMapper.Mapper.Map<Reservation,ReservationEntity>
(entity);
        }
        public Reservation GetDataEntity(ReservationEntity entity)
        {
            return AutoMapper.Mapper.Map<ReservationEntity,Reservation>
(entity);
        }
    }
     /// <summary>
    /// 预约表单字段
    /// </summary>
    [Serializable]
    public class ReservationContentEntity : IviewModel
<ReservationContentEntity, ReservationContent>
    {
        static ReservationContentEntity()
        {
            AutoMapper.Mapper.CreateMap<ReservationContent,
ReservationContentEntity>();
            AutoMapper.Mapper.CreateMap<ReservationContentEntity,
ReservationContent>();
        }

        /// <summary>
        /// ID
        /// </summary>
        [Key]
        public int ID { get; set; }
        /// <summary>
        /// 字段名称
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// 初始内容
        /// </summary>
        public string Content { get; set; }
        /// <summary>
        /// 预约表单 ID
        /// </summary>
        public int ReservationID { get; set; }
```

```
        public  ReservationContentEntity  GetViewModel(ReservationContent
entity)
        {
            return AutoMapper.Mapper.Map<ReservationContent,
ReservationContentEntity> (entity);
        }
        public ReservationContent GetDataEntity(ReservationContentEntity
entity)
        {
            return AutoMapper.Mapper.Map<ReservationContentEntity,
ReservationContent> (entity);
        }
    }
    /// <summary>
    /// 预约填表用户
    /// </summary>
    public class UserReservationEntity : IViewModel<UserReservationEntity,
UserReservation>
    {
        static UserReservationEntity()
        {
            AutoMapper.Mapper.CreateMap<UserReservation,
UserReservationEntity>();
            AutoMapper.Mapper.CreateMap<UserReservationEntity,
UserReservation>();
        }
        /// <summary>
        /// ID
        /// </summary>
        [Key]
        public int ID { get; set; }

        /// <summary>
        /// 微信个人用户 OpenId
        /// </summary>
        public string WeixinOpenId { get; set; }
        /// <summary>
        /// 预约表单 ID
        /// </summary>
        public int ReservationID { get; set; }
        public UserReservationEntity GetViewModel(UserReservation entity)
```

```
            {
              return AutoMapper.Mapper.Map<UserReservation, UserReservationEntity>
(entity);
            }
            public UserReservation GetDataEntity(UserReservationEntity entity)
            {
              return AutoMapper.Mapper.Map<UserReservationEntity, UserReservation>
(entity);
            }
        }
        /// <summary>
        /// 用户填写的表单信息
        /// </summary>
        public class UserReservationContentEntity : IViewModel
<UserReservationContentEntity, UserReservationContent>
        {
            static UserReservationContentEntity()
            {
                AutoMapper.Mapper.CreateMap<UserReservationContent,
UserReservationContentEntity>();
                AutoMapper.Mapper.CreateMap<UserReservationContentEntity,
UserReservationContent>();
            }
            /// <summary>
            /// ID
            /// </summary>
            [Key]
            public int ID { get; set; }
            /// <summary>
            /// 字段填写内容
            /// </summary>
            public string Content { get; set; }
            /// <summary>
            /// 填表用户 ID
            /// </summary>
            public int UserReservationId { get; set; }
            /// <summary>
            /// 预约表单字段 ID
            /// </summary>
            public int ReservationContentId { get; set; }
```

```
        public UserReservationContentEntity GetViewModel
(UserReservationContent entity)
        {
            return AutoMapper.Mapper.Map<UserReservationContent,
UserReservationContentEntity>(entity);
        }
        public UserReservationContent GetDataEntity(UserReservationContentEntity
entity)
        {
            return AutoMapper.Mapper.Map<UserReservationContentEntity,
UserReservationContent>(entity);
        }
    }
}
```

6.3.6　实现业务逻辑层

由于我们采用“改列为行，用另外一个表存放定制字段”的解决方案来实现通用预约表单的设计，因此一个完整的预约表单信息被分别存放在 Reservation 及 ReservationContent 等两张表中，用户填写的预约信息同样被分别存放在 UserReservation 及 UserReservationContent 等两张表中。新增预约表单信息及新增用户填写的预约信息需要使用事务（TransactionScope）来确保数据的一致性。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Transactions;
using System.Web;
using Sample_3.ORM;

namespace Sample_3
{
    public class ReservationBll
    {
        /// <summary>
        /// 本演示程序仅保存一个预约表单
        /// </summary>
        const int ID = 1;
        /// <summary>
        /// 获取预约实体
```

```
        /// </summary>
        /// <returns></returns>
        public ReservationEntity GetReservation()
        {
            var dataEntity = new ReservationDll().LoadEntity(p => p.ID == ID);
            return new ReservationEntity().GetViewModel(dataEntity);
        }
        /// <summary>
        /// 添加预约表单及预约表单字段
        /// </summary>
        /// <param name="name"></param>
        /// <returns></returns>
        public bool AddReservation(string name, List<ReservationContentEntity>
viewFormRows)
        {
            var viewEntity = new ReservationEntity() { Name = name };
            try
            {
              using (TransactionScope transactionScope = new TransactionScope())
                {
                    //添加预约表单
                    var dataEntity = viewEntity.GetDataEntity(viewEntity);
                    int id = new ReservationDll().AddReservation(dataEntity);
                    if (id > 0)
                    {
                        //添加预约表单字段
                        viewFormRows.ForEach(p => p.ReservationID = id);
                        List<ReservationContent> dataFormRow = viewFormRows.
Select(p => p.GetDataEntity(p)).ToList();
                        if (new ReservationContentDll().AddReservationContent
(dataFormRow))
                        {
                            //如果添加预约表单及预约表单字段全部成功，则提交事务
                            transactionScope.Complete();
                            return true;
                        }
                    }
                }
            }
            catch { }
            return false;
```

```
        }
    }
    public class ReservationContentBll
    {
        /// <summary>
        /// 获取预约表单字段实体
        /// </summary>
        /// <param name="reservationID">预约 ID</param>
        /// <returns></returns>
        public List<ReservationContentEntity> GetReservationContents(int
reservationID)
        {
            var dataEntities = new ReservationContentDll().LoadEntities(p =>
p.ReservationID == reservationID).ToList();
            //转换为 ViewEntity
            var viewEntity = new ReservationContentEntity();
            return
dataEntities.Select(p=>viewEntity.GetViewModel(p)).ToList();
        }
    }
    public class UserReservationBll
    {
        /// <summary>
        /// 添加用户填写的预约表单及预约表单字段内容
        /// </summary>
        /// <param name="openId">微信 OpenId</param>
        /// <param name="reservationID">预约 ID</param>
        /// <returns></returns>
        public bool AddUserReservation(string openId, int reservationID,
List<UserReservationContentEntity> viewFormRows)
        {
            var viewEntity = new UserReservationEntity() { ReservationID =
reservationID, WeixinOpenId = openId };
            try
            {
             using (TransactionScope transactionScope = new TransactionScope())
               {
                  //添加用户填写的预约表单
                  var dataEntity = viewEntity.GetDataEntity(viewEntity);
                 int id = new UserReservationDll().AddEntity(dataEntity).ID;
                  if (id > 0)
```

```
                    {
                        //添加用户填写的预约表单字段内容
                        viewFormRows.ForEach(p => p.UserReservationId = id);
                        List<UserReservationContent> dataFormRow =
viewFormRows.Select(p => p.GetDataEntity(p)).ToList();
                        if (new UserReservationContentDll().
AddUserReservationContent(dataFormRow))
                        {
                            //如果添加用户填写的预约表单及预约表单字段内容全部成功，则
提交事务
                            transactionScope.Complete();
                            return true;
                        }
                    }
                }
            }
            catch { }
            return false;
        }
    }
}
```

6.4 预约系统实现

6.4.1 预约系统后台实现

预约系统后台提供设计预约表单的功能供微信公众号使用。

在项目中新建一个名为 Admin.aspx 的 Web 窗体，作为预约表单设计页面。这里使用了 Bootstrap 作为前端开发框架。Bootstrap 是 Twitter 推出的一个用于前端开发的开源工具包。感兴趣的朋友可以到 Bootstrap 中文网：http://www.bootcss.com/，详细了解 Bootstrap。

为了能动态地添加和删除表单的行，我们使用了 Jquery 的插件 Jquery Template Plugin。Jquery Template Plugin 是一个使用方便的 HTML 模版引擎，其官方网址为：https://github.com/BorisMoore/jquery-tmpl。

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Admin.aspx.cs"
Inherits="Sample_3.Admin" %>
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
       <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
       <title>预约表单设计</title>
    </head>
    <!-- #Bootstrap -->
    <link href="Css/bootstrap.min.css" rel="stylesheet" />
    <script src="Scripts/bootstrap.min.js"></script>
    <!-- #Jquery -->
    <script src="Scripts/jquery-1.9.1.min.js"></script>
    <!-- #Jquery Template -->
    <script src="Scripts/jquery.tmpl.min.js"></script>
    <body>
       <div class="container">
          <form class="form-horizontal" type="post" runat="server">
             <input type="hidden" id="lineCount" name="lineCount" value="">
             <h2 class="form-signin-heading">预约表单设计</h2>
             <%-- 表单名称输入框 --%>
             <div class="control-group">
                <label class="control-label" for="formName"><strong>表单名称
</strong></label>
                <div class="controls">
                   <input type="text" name="formName" value=""/>
                </div>
             </div>
             <%-- 表单字段设计部分 --%>
             <table id="listTable" class="table table-bordered table-hover
dataTable" style="width: auto">
                <thead>
                   <tr>
                      <th style="border-color: #ddd; color: Black">字段名
                      </th>
                      <th style="border-color: #ddd; color: Black">字段名称
                      </th>
                      <th style="border-color: #ddd; color: Black">初始内容
                      </th>
                      <th style="border-color: #ddd; color: Black">字段类型
                      </th>
```

```
                        <th style="border-color: #ddd; color: Black">操作
                        </th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <%-- 提交按钮 --%>
            <button class="btn btn-large btn-primary" type="submit">保存
</button>
        </form>
    </div>
</body>
<%-- 一行表单字段模版--%>
<script type="text/x-jquery-tmpl" id="replyOrderlist">
    <tr>
        <td>字段:
        </td>
        <td>
            <input type="text" name="Name${Index}" onkeyup="value=value.
substring(0,20)" value="">
        </td>
        <td>
            <input name="Content${Index}" type="text" onkeyup="value=value.
substring(0,500)" value="">
        </td>
        <td>文本框
        </td>
        <td>
            <p>
                <a class="btnGrayS vm doaddit" href="javascript:void(0)">添
加</a> <a href='javascript:void(0)' class='dodelit'>删除</a>
            </p>
        </td>
    </tr>
</script>
<script type="text/javascript">
    var count = 0;
    //生成一行表单字段
    function createLine() {
        var line = {
```

```
            Index: count
        };
        count++;
        $("#lineCount").val(count);
        return $("#replyOrderlist").tmpl(line);
    }
    //页面加载完毕初始化第一行
    $(function () {
        createLine().appendTo("#listTable");
    });
    //注册删除按钮事件
    $(document).on("click", ".dodelit", function () {
        $(this).parent().parent().parent().remove();
    });
    //注册添加按钮事件
    $(document).on("click", ".doaddit", function () {
        $(this).parent().parent().parent().after(createLine());
    });
</script>
</html>
```

页面提交数据的处理在 Page_Load 中实现。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace Sample_3
{
    public partial class Admin : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (Page.IsPostBack)
            {
                //表单 POST 提交操作
                int count;
                //获取表单行数
                int.TryParse(Request.Form["lineCount"], out count);
                //获取预约名称
```

```
                string reservationName = Request.Form["formName"];
                if (count > 0 && !string.IsNullOrEmpty(reservationName))
                {
                    //从 POST 提交的数据，构造预约表单字段实体
                    List<ReservationContentEntity> formRow = new List
<ReservationContentEntity>();
                    for (int i = 0; i < count; ++i)
                    {
                        string name = Request.Form["Name" + i.ToString()];
                     string content = Request.Form["Content" + i.ToString()];
                     if (!string.IsNullOrEmpty(name) && !string.IsNullOrEmpty
(content))
                        {
                            formRow.Add(new ReservationContentEntity() { Name
= name, Content = Request.Form["Content" + i.ToString()] });
                        }
                    }
                    if (formRow.Count > 0)
                    {
                        //添加预约表单及预约表单字段
                     if (new ReservationBll().AddReservation(reservationName,
formRow))
                        {
                            Response.Write("保存成功");
                            Response.End();
                        }
                        else
                        {
                            Response.Write("保存失败");
                            Response.End();
                        }
                    }
                }
            }
        }
    }
```

6.4.2　预约系统前台实现

预约系统前台提供预约表单填写的功能供微信个人用户使用。

在项目中新建一个名为 Form.aspx 的 Web 窗体，作为预约表单填写页面。这里使用了 Jquery Mobile 作为前端开发框架。Jquery Mobile 是创建移动 Web 应用程序的前端框架，使用该框架可以快速建立供手机浏览器显示的 HTML 5 页面。Jquery Mobile 的中文网站：http://www.jqmapi.com 中有关于 Jquery Mobile 的详细介绍。

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Form.aspx.cs"
Inherits="Sample_3.Form" %>

    <!DOCTYPE html>
    <html>
    <head>
        <meta content="text/html; charset=utf-8" http-equiv="Content-Type" />
        <!-- #手机浏览器兼容性设置 -->
        <meta content="application/xhtml+xml;charset=UTF-8" http-equiv="Content-
Type">
        <meta content="no-cache,must-revalidate" http-equiv="Cache-Control">
        <meta content="no-cache" http-equiv="pragma">
        <meta content="0" http-equiv="expires">
        <meta content="telephone=no, address=no" name="format-detection">
        <meta        content="width=device-width,        initial-scale=1.0"
name="viewport">
        <meta name="apple-mobile-web-app-capable" content="yes" />
        <meta name="apple-mobile-web-app-status-bar-style" content="black-
translucent"/>
        <!-- #手机浏览器兼容性设置 -->
        <title>预约</title>
        <!-- #Jquery -->
        <script src="Scripts/jquery-1.9.1.min.js"></script>
        <!-- #Jquery Template -->
        <script src="Scripts/jquery.tmpl.min.js"></script>
        <!-- #Jquery Mobile -->
        <script src="Scripts/jquery.mobile-1.3.2.min.js"></script>
        <link href="Css/jquery.mobile-1.3.2.min.css" rel="stylesheet" />
        </head>
        <body>
            <%-- 预约表单名称 --%>
```

```
            <h3><%=(ViewState["Reservation"] as Sample_3.ReservationEntity).
Name%></h3>
            <%-- 根据预约表单设计生成用户需要填写的表单 --%>
            <form      id="Form1"      class="form-horizontal"      type="post"
runat="server">
              <input type="hidden" name="reservationID" id="reservationID" value
="<%=(ViewState["Reservation"] as Sample_3.ReservationEntity).ID%>" />
               <input    type="hidden"    name="openID"    id="openID"    value
="<%=Request.QueryString["openId"]%>" />
            <% foreach (Sample_3.ReservationContentEntity content in (ViewState
["ReservationContents"] as List<Sample_3.ReservationContentEntity>))
               { %>
            <li data-role='fieldcontain'>
               <label for='content<%= content.ID%>'><%= content.Name%>: </label>
               <input type='text' placeholder='<%= content.Content%>' name='
content<%= content. ID%>' id='content<%= content.ID%>' /></li>
            <% } %>
            <%-- 表单提交按钮 --%>
            <input type="submit" data-theme="b" value="提交" id="bsubmit" rel=
"external"/>
            </form>
         </body>
      </html>
```

表单初始化及页面提交数据的处理在 Page_Load 中实现。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace Sample_3
{
   public partial class Form : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (Page.IsPostBack)
         {
            //表单 POST 提交操作
            //获取预约表单 ID
```

```
                int reservationID;
                int.TryParse(Request.Form["reservationID"], out reservationID);
                if (reservationID > 0)
                {
                    //获取微信个人用户的 OpenID
                    string openId = Request.Form["openId"];
                    if (!string.IsNullOrEmpty(openId))
                    {
                        //从 POST 提交的数据，构造用户填写的预约表单各字段的内容实体
                        List<UserReservationContentEntity> formRow = new List<
UserReservationContentEntity>();
                        var reservationContents = ReservationContents
(reservationID);
                        foreach (var reservationContent in reservationContents)
                        {
                            string content = Request.Form["content" +
reservationContent.ID.ToString()];
                            formRow.Add(new UserReservationContentEntity()
{ Content = content, ReservationContentId = reservationContent.ID });
                        }
                        if (formRow.Count > 0)
                        {
                            //添加用户填写的预约表单及预约表单字段内容
                            if (new UserReservationBll().AddUserReservation
(openId, reservationID, formRow))
                            {
                                Response.Write("保存成功");
                                Response.End();
                            }
                            else
                            {
                                Response.Write("保存失败");
                                Response.End();
                            }
                        }
                    }
                }
                else
                {
                    //页面初始化，获取前端页面需要的预约表单信息
```

```
                var reservation = Reservation();
                ViewState["Reservation"] = reservation;
                ViewState["ReservationContents"] = ReservationContents
(reservation.ID);
            }
        }
        /// <summary>
        /// 获取预约实体
        /// </summary>
        /// <returns></returns>
        protected ReservationEntity Reservation()
        {
            return new ReservationBll().GetReservation();
        }
        /// <summary>
        /// 获取预约表单字段实体
        /// </summary>
        /// <param name="reservationID">预约 ID</param>
        /// <returns></returns>
        protected List<ReservationContentEntity> ReservationContents(int
reservationID)
        {
            return new ReservationContentBll().GetReservationContents
(reservationID);
        }
    }
}
```

6.5 部署及测试体验

在Sample_3项目本地调试通过后，需要将项目上传到AppHarbor中进行部署，以便让微信服务器能访问到我们的程序，进行实际测试。AppHarbor 部署的详细步骤在第 4 章有完整讲解。

在将 Sample_3 项目部署到 AppHarbor 后，需要先访问 admin.aspx 进行预约表单设计，如图 6-6 所示。

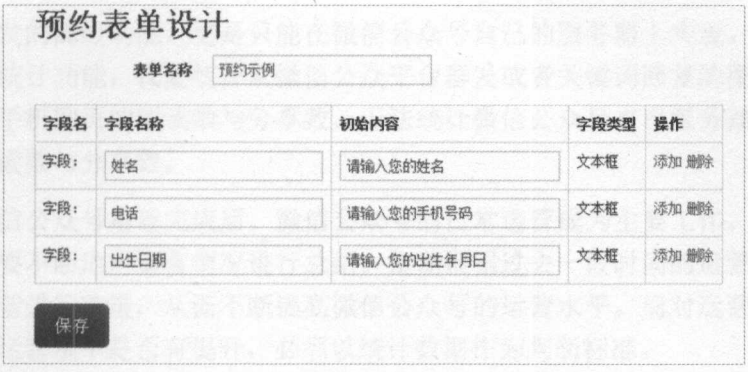


图 6-6

然后使用微信客户端关注微信公众号。向该微信公众号发送“预约测试”，单击回复的单图文，进入预约表单填写页面，如图 6-7 所示。

填写完预约表单后，单击“提交”按钮，用户的预约信息将被保存到数据库中。

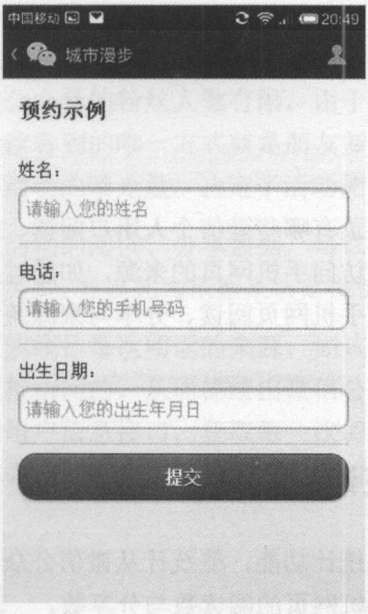


图 6-7

第 7 章
商用案例 2——阅读、分享统计

本章主要讲解如何记录有哪些微信个人用户阅读、分享了微信公众号的手机网页，以及微信个人用户访问手机网页的来源，如通过朋友圈分享访问、通过好友分享消息访问等；实现手机网页阅读、分享与来源统计及手机网页在朋友圈的传播路径分析。

7.1 阅读、分享统计的意义

微信公众平台自带的统计功能，能统计从微信公众平台群发或者关键词回复的图文消息及图文详情手机网页的阅读数与分享数。

微信公众平台只有简单的功能，如果微信公众号要实现相对复杂的功能，则必须使用开发者模式。使用开发者模式后，群发消息、关键词回复等功能都是在微信公众号自己的服务器上实现的。同时，像微信官网、预约、活动等微信公众

号自己开发的高级功能，更是只能在微信公众号自己的服务器上实现。而微信公众平台的统计功能，仅能统计从微信公众平台群发或者关键词回复的图文消息及图文详情手机网页的阅读数与分享数，无法统计微信公众号自己服务器上的手机网页的阅读数与分享数。

当微信公众号搭建完成后，微信公众号的日常运营成为主要工作。在运营过程中，需要不断地对运营情况进行总结。定期根据过去一段时间的运营情况，对后续的运营进行改进，从而不断提高微信公众号的运营水平。而对运营结果好坏的评判，运营水平是否有提升，必须以统计数据作为判断标准。

最基本、最重要的统计数据，就是每一个手机网页被微信个人用户阅读的次数及被分享到朋友圈的次数。阅读次数代表微信公众号的影响力(即受众的多寡)，以及一篇内容性文章或一次活动的传播广度。被分享到朋友圈的次数则表明了一篇内容性文章内容质量的好坏，只有拥有高质量的内容，引起阅读者的认同、共鸣或者八卦、猎奇的心理，一篇文章才会被微信个人用户转发到朋友圈，分享给他的微信好友。而对微信公众号举行的一次活动，被分享到朋友圈的次数，直接表明了该次活动的策划成功与否。微信公众号举办的活动，总是希望能有更多的人参与，目前，一般微信公众号的粉丝人数有限，由于微信产品本身的密友社交属性，一个活动能被非粉丝看到的唯一方式就是朋友圈分享。因此，被分享到朋友圈的次数，直接代表了活动的曝光量，决定了活动能有多少人参与。

通过对比不同文章或活动的阅读、分享数据，可以找出最受粉丝欢迎的文章类型及参与度最高的活动类型，并以此为依据来决定微信公众号的后续运营策略。

更进一步的，我们还能统计每次阅读的来源，如该次阅读是通过哪个微信个人用户分享到朋友圈的链接访问的，从而描绘出微信公众号手机网页的传播路径图。利用微信公众平台的用户信息接口，获取每一次阅读的微信个人用户基本信息，对受众进行分组与画像，更好地指导微信公众号的运营工作。

7.2　获取分享记录

7.2.1　微信 JS 接口简介

通过微信公众平台群发功能发送的消息，在图文详情页面中单击标题下方的

微信公众号名称，可以直接关注该微信公众号。群发消息的图文详情是一个手机网页，而网页的任何功能性操作，只能通过 JavaScript 来实现。通过阅读微信公众平台开发者文档中“Weixin JS 接口”部分的内容及分析微信程序，发现微信内嵌浏览器有一个私有 JavaScript 对象 WeixinJSBridge，通过操作这个对象的相关方法可以实现捕获转发链接到微信朋友圈、转发链接给微信好友、转发链接到腾讯微博、转发链接到 Facebook 和判断一个微信号的关注状态等。

以上这些功能都需要在微信内嵌浏览器内才能被识别，通过判断 WeixinJSBridge 对象是否存在，我们可以知道页面是否在微信内嵌浏览器中被打开。

WeixinJSBridge.on 方法可以注册当微信个人用户单击微信右上角“...”菜单中的发送给朋友、分享到朋友圈、分享到腾讯微博等功能时的事件触发函数。

WeixinJSBridge.invkoe 方法则实现在事件触发后，调用 WeixinJSBridge 中的各类分享函数，并在分享动作结束后，调用自定义的回调函数，以判断分享是否成功。

7.2.2 使用微信 JS 接口获取分享记录

为了能方便地在任何页面中记录发送给朋友、分享到朋友圈等动作，需要建立一个 JavaScript 文件 StatisticsWeChatAPI.js。所有借用 WeixinJSBridge 接口，捕获发送给朋友、分享到朋友圈等动作的相关 JavaScript 代码都集中在该文件中。任意网页只需要引用 StatisticsWeChatAPI.js 文件，就能实现记录发送给朋友、分享到朋友圈等动作的功能。

StatisticsWeChatAPI.js 的完整代码如下：

```
    //记录发送给朋友、分享到朋友圈等动作的默认配置，在实际记录分享动作的页面中赋以真实的配置值
    var dataForWeixin = {
        appId: "xxxxxxxxx",
        MsgImg: "转发时的图片",
        TLImg: "图片",
        url: "自定义链接",
        title: "自定义标题",
        desc: "自定义描述",
        fakeid: "",
        //发送给朋友成功后的默认回调函数
```

```
        //在此回调函数中，将分享记录发送到服务器中保存
        friendcallback: function () {
            alert("分享成功！");
        },
        //分享到朋友圈成功后的默认回调函数
        //在此回调函数中，将分享记录发送到服务器中保存
        friendCirclecallback: function () {
            alert("分享成功！");
        }
    };
    (function () {
        //微信内置浏览器加载完毕后，当粉丝将网页发送给朋友或分享到朋友圈时，触发的事件处理
函数
        var onBridgeReady = function () {
            //发送给朋友
            WeixinJSBridge.on('menu:share:appmessage', function (argv) {
                WeixinJSBridge.invoke('sendAppMessage', {
                    "appid": dataForWeixin.appId,
                    "img_url": dataForWeixin.MsgImg,
                    "img_width": "120",
                    "img_height": "120",
                    "link": dataForWeixin.url,
                    "desc": dataForWeixin.desc,
                    "title": dataForWeixin.title
                }, function (res) {
                    alert(res.err_msg);
                    if (res.err_msg == "send_app_msg:ok" || res.err_msg ==
"send_app_msg:confirm") {
                        dataForWeixin.friendcallback();
                    }
                });
            });
            //发送到朋友圈
            WeixinJSBridge.on('menu:share:timeline', function (argv) {
                //安卓系统无法触发回调，单独处理
                if (IsAndroid()) {
                    dataForWeixin.friendCirclecallback();
                    WeixinJSBridge.invoke('shareTimeline', {
                        "img_url": dataForWeixin.TLImg,
                        "img_width": "120",
                        "img_height": "120",
```

```
                    "link": dataForWeixin.url,
                    "desc": dataForWeixin.desc,
                    "title": dataForWeixin.title
                }, function (res) {
                    if (res.err_msg == 'share_timeline:ok' || res.err_msg ==
'share_timeline:confirm') {
                        // dataForWeixin.friendCirclecallback();
                    }
                });
            } else {
                WeixinJSBridge.invoke('shareTimeline', {
                    "img_url": dataForWeixin.TLImg,
                    "img_width": "120",
                    "img_height": "120",
                    "link": dataForWeixin.url,
                    "desc": dataForWeixin.desc,
                    "title": dataForWeixin.title
                }, function (res) {
                    if (res.err_msg == 'share_timeline:ok' || res.err_msg ==
'share_timeline:confirm') {
                        dataForWeixin.friendCirclecallback();
                    }
                });
            }
        });
        //分享到微博
        WeixinJSBridge.on('menu:share:weibo', function (argv) {
            WeixinJSBridge.invoke('shareWeibo', {
                "content": dataForWeixin.title,
                "url": dataForWeixin.url
            }, function (res) {
                if (res.err_msg == "share_weibo:ok") {
                    dataForWeixin.callback();
                }
            });
        });
        //分享到 Facebook
        WeixinJSBridge.on('menu:share:facebook', function (argv) {
            (dataForWeixin.callback)();
            WeixinJSBridge.invoke('shareFB', {
                "img_url": dataForWeixin.TLImg,
```

```
                "img_width": "120",
                "img_height": "120",
                "link": dataForWeixin.url,
                "desc": dataForWeixin.desc,
                "title": dataForWeixin.title
            }, function (res) { });
        });
        //判断手机操作系统是否为安卓系统
        function IsAndroid() {
            var userAgentInfo = navigator.userAgent;
            if (userAgentInfo.indexOf("Android") > 0) { return true; }
            return false;
        }
    };
    //判断网页是否从微信内置浏览器打开
    //如果网页是在微信内置浏览器中打开的，则注册记录发送给朋友、分享到朋友圈等动作的事
件触发函数
    if (typeof WeixinJSBridge == "undefined") {
        if (document.addEventListener) {
            document.addEventListener('WeixinJSBridgeReady', onBridgeReady,
false);
        } else if (document.attachEvent) {
            document.attachEvent('WeixinJSBridgeReady', onBridgeReady);
            document.attachEvent('onWeixinJSBridgeReady', onBridgeReady);
        }
    }else{
        onBridgeReady();
    }
})();
```

7.3　获取访问来源

在微信中，微信个人用户只能通过以下几种方式打开微信公众号自己服务器的手机网页：

1. 单击微信公众号群发消息的图文详情页中的“查看原文”链接。
2. 单击微信公众号自定义菜单中的网页链接类型的按钮。
3. 发送消息给微信公众号，单击微信公众号回复的图文消息。

4．单击微信好友通过“发送给朋友”按钮分享链接。

5．单击微信朋友圈中朋友分享的链接。

其中，通过前三种方式打开手机网页，我们可以统一定义为通过“直接在微信公众号中打开微信内置浏览器”的方式访问。

因此，可以通过一个枚举 NavFrom，来定义一个手机页面的访问来源类型：

```
/// <summary>
/// 访问来源类型
/// </summary>
public enum NavFrom
{
    /// <summary>
    /// 微信朋友圈
    /// </summary>
    Timeline,
    /// <summary>
    /// 微信好友发送的链接
    /// </summary>
    Message,
    /// <summary>
    /// 直接在微信公众号中打开微信浏览器
    /// </summary>
    MicroMessenger,
    /// <summary>
    /// 其他（不是在微信中访问）
    /// </summary>
    Other
}
```

分享到朋友圈的网页链接，微信会自动在网页链接的末尾加上字符串“&from=timeline”，以标识该链接是被分享到朋友圈的链接。

发送给朋友的网页链接，微信会自动在网页链接的末尾加上字符串“&from=Message”，以标识该链接是微信好友发送的链接。

所有来自于微信内置浏览器的访问请求，HTTP 请求头部的“User-Agent”参数，都会由微信自动加上字符串“MicroMessenger”。通过判断 HTTP 请求头部的“User-Agent”参数是否包含“MicroMessenger”，就可以判断一个页面访问请求是

否来自于微信。

通过以上分析，只需要一个简单的方法 GetNavFromType，就可以完整地判断访问微信公众号服务器上手机网页的来源类型。

```
    /// <summary>
    /// 判断页面访问来源类型
    /// </summary>
    /// <returns></returns>
    private static NavFrom GetNavFromType()
    {
        //网址中的参数集合
        NameValueCollection parameters = System.Web.HttpContext.Current.Request.
Params;
        string fromStr = parameters["from"]; //发送给朋友、分享到朋友圈的链接会含
有 from 参数
        NavFrom fromType;
        if (!Enum.TryParse<NavFrom>(fromStr, true, out fromType)) //通过判断
from 参数，识别页面访问是来自于发送给朋友的链接还是分享到朋友圈的链接
        {
            //获取 HTTP 访问头中的 User-Agent 参数的值
            string agent = System.Web.HttpContext.Current.Request.Headers
["User-Agent"];
            if (agent.Contains(NavFrom.MicroMessenger.ToString())) //判断页面是
否是在微信内置浏览器中打开
                fromType = NavFrom.MicroMessenger;
            else
                fromType = NavFrom.Other;
        }
        return fromType;
    }
```

7.4　识别访问者与分享者

7.4.1　识别访问者

通过微信公众平台提供的“网页授权接口”，可以方便地获取访问网页的微信个人用户的 OpenId。由于每个微信个人用户的 OpenId 的唯一性，我们可以以

OpenId 作为访问者的特征标识，来识别访问者。在第 5 章的 5.8 节中，有对微信公众平台“网页授权接口”的详细介绍。

要获取微信个人用户的 OpenId，每一次页面请求都需要执行微信公众平台“网页授权接口”前两个步骤。这两个步骤会执行两次页面跳转，第一次是从微信个人用户访问的微信公众号服务器页面跳转到微信公众平台服务器获取网页授权 Code 的页面。第二次是微信公众平台服务器重定向请求到微信个人用户访问的微信公众号服务器页面并附加上 Code 值。微信公众号服务器在获取到 Code 后，再次访问“网页授权接口”，以获取访问网页的微信个人用户的 OpenId。

如果每次页面请求都访问微信公众平台的“网页授权接口”获取微信个人用户的 OpenId，则必然导致页面打开速度变慢，降低用户体验，同时也会加重微信公众号服务器的负担。

因此，有必要在第一次获取到微信个人用户的 OpenId 后，将微信个人用户的 OpenId 保存在 Cookie 中。每次页面请求，如果 Cookie 中存在微信个人用户的 OpenId，则直接从 Cookie 中读取，以识别访问者。

7.4.2 识别分享者

当微信个人用户单击微信好友通过“发送给朋友”按钮分享的链接，或单击微信朋友圈中朋友分享的链接，来访问微信公众号的手机页面时，除获取访问来源与识别访问者外，我们还需要识别该次访问是通过哪个微信个人用户分享的链接访问的，即识别分享者，从而为描绘出微信公众号手机网页的传播路径图提供数据基础。

一种简单实现识别分享者的方式，是在页面网址中加上当前访问页面的微信个人用户的 OpenId。如果这个微信个人用户分享了页面，则在分享的链接地址中会含有分享人的 OpenId。当这个微信个人用户的微信好友单击了分享链接地址，通过“网页授权接口”获取到的访问者的 OpenId 或 Cookie 保存的访问者的 OpenId 与页面网址中的 OpenId 会出现不一致的情况。当发现页面网址中的 OpenId 与实际获取到当前访问者的 OpenId 不同时，页面网址中的 OpenId 即为分享者的 OpenId。在记录分享者信息后，将页面网址中的 OpenId 替换为当前访问者的 OpenId，以确保当前访问者分享页面链接后，同样能识别正确的分享者信息。

7.4.3　实现识别访问者与分享者

按照前面两个小节的分析思路，要实现识别访问者与分享者，首先需要建立一个保存和读取 Cookie 的类 CookieHelper。

```
    /// <summary>
    /// 操作站内 Cookie 的助手
    /// </summary>
    public class CookieHelper
    {
        /// <summary>
        /// 写客户端 Cookie 的名字
        /// </summary>
        public const string COOKIE_NAME = "Sample_4";
        #region 写 Cookie 到客户端
        /// <summary>
        /// 登录后写 Cookie 到客户端,代替 session
        /// </summary>
        /// <param name="expires">过期时间，如果永不过期，则设为 DateTime.MaxValue，
<para>如果不想写入客户端，浏览器关闭时即失效，则设为 DateTime.MinValue</para></param>
        /// <param name="values">保存 Cookie 信息</param>
        public static void WriteLoginCookies(Dictionary<string, string> values,
DateTime expires)
        {
          HttpCookie cookie = HttpContext.Current. Request.Cookies[COOKIE_NAME] ??
new HttpCookie(COOKIE_NAME);
            if (expires != DateTime.MinValue)
               cookie.Expires = expires;
            foreach (var value in values)
            {
              cookie.Values[value.Key] = System.Web.HttpUtility.UrlEncode(value.
Value);
            }
            HttpContext.Current.Response.Cookies.Add(cookie);
        }
        #endregion
        #region 从 Cookie 中获取信息
        /// <summary>
        /// 从 Cookie 中获取信息;
        /// </summary>
        /// <returns></returns>
```

```
        public static Dictionary<string, string> GetLoginCookies(string[]
cookieKeys)
        {
            Dictionary<string, string> values = new Dictionary<string,
string>();
            HttpCookie cookie = HttpContext.Current.Request.Cookies[COOKIE_NAME];
            if (cookie != null)
            {
                foreach (var cookieKey in cookieKeys)
                {
                    string value = cookie.Values[cookieKey];
                    values.Add(cookieKey,
                            value == null ? null : System.Web.HttpUtility.UrlDecode
(cookie.Values[cookieKey].Trim()).Replace("%5F", "_"));
                }
            }
            else
            {
                foreach (var cookieKey in cookieKeys)
                {
                    values.Add(cookieKey, null);
                }
            }
            return values;
        }
        #endregion
        #region 清除Cookie
        /// <summary>
        /// 清除Cookie
        /// </summary>
        public static void CleanLoginCookie(string[] keys)
        {
            HttpCookie ck = HttpContext.Current.Request.Cookies[COOKIE_NAME];

            if (ck == null || ck.Values.Count == 0)
                return;
            foreach (var key in keys)
            {
                ck.Values[key] = "";
            }
            ck.Expires = DateTime.Now.AddDays(-1.0);
```

```
            HttpContext.Current.Response.Cookies.Add(ck);
        }
        #endregion
    }
```

获取微信个人用户的 OpenId 以识别访问者，以及识别分享者的代码如下：

```
    /// <summary>
    /// 获取访问者 OpenId
    /// </summary>
    private string GetNavOpenId()
    {
        NameValueCollection parameters = System.Web.HttpContext.Current.Request.
Params;
        //获取链接中的 OpenId
        string navOpenId = parameters["u"];
        #region 如果是从微信浏览器浏览，则可获取真实的微信 OpenId
        if (!string.IsNullOrEmpty(appID) && !string.IsNullOrEmpty(appsecret))
        {
            string accessSource = System.Web.HttpContext.Current.Request.
ServerVariables["HTTP_USER_AGENT"];
            if (accessSource.Contains("MicroMessenger")) //如果是从微信打开页面
            {
                string[] cookieKeys = new[] { CookieHelper.COOKIE_NAME };
                Dictionary<string, string> realIdCookie = CookieHelper.
GetLoginCookies(cookieKeys);//获取保存在 Cookie 中的 OpenId
                //如果 Cookie 中不存在 OpenId，或者链接中的 OpenId 与 Cookie 中的 OpenId
不一致，则链接中的 OpenId 为分享者的 OpenId，需要获取当前用户的真实 OpenId
                if (NeedGetReadOpenId(parameters, realIdCookie))
                {
                    if (parameters["code"] == null)
                    {
                        // 先去获取 Code，并记录分享者
                        string snsapi_baseUrl = GoCodeUrl(navOpenId);
                        if (!string.IsNullOrEmpty(snsapi_baseUrl))
                        {
                            CookieHelper.CleanLoginCookie(cookieKeys);
                            //跳转到微信网页授权页面

System.Web.HttpContext.Current.Response.Redirect(snsapi_baseUrl, true);
                            System.Web.HttpContext.Current.Response.End();
                            return null;
```

```
                    }
                }
                else
                {
                 OAuthAccessTokenResult tokenResult = GetRealOpenId(parameters
["code"].ToString());
                  if (null != tokenResult && !string.IsNullOrEmpty(tokenResult.
openid))
                    {
                        navOpenId = tokenResult.openid;
                        // 获取到的当前访问者的 OpenId 保存到 Cookie 里
                        CookieHelper.CleanLoginCookie(cookieKeys);
                        realIdCookie[CookieHelper.COOKIE_NAME] = tokenResult.
openid;
                       CookieHelper.WriteLoginCookies(realIdCookie, DateTime.
MinValue);
                    }
                }
            }
        }
        #endregion
        return navOpenId;
    }
    /// <summary>
    /// 如果 Cookie 中存在 OpenId 且链接中的 OpenId 与 Cookie 中的 OpenId 一致
    /// 则不需要调用网页授权接口，链接中的 OpenId 即为当前访问者的真实 OpenId
    /// </summary>
    /// <param name="parameters"></param>
    /// <param name="realIdCookie"></param>
    /// <returns></returns>
    private bool NeedGetReadOpenId(NameValueCollection parameters, Dictionary<
string, string> realIdCookie)
    {
        string referer = System.Web.HttpContext.Current.Request.
ServerVariables["HTTP_REFERER"];
        string openId = null;
        if (realIdCookie != null)
        {
            if (realIdCookie.ContainsKey(CookieHelper.COOKIE_NAME))
            {
```

```
            openId = realIdCookie[CookieHelper.COOKIE_NAME];
        }
    }
    if (!string.IsNullOrEmpty(referer) && openId == parameters["u"].ToString())
        return false;
    else
        return true;
}
/// <summary>
/// 网页授权接口第一步
/// 跳转到获取 Code 的 URL
/// </summary>
/// <param name="shareOpenId">当访问来源为朋友圈时的分享者微信 OpenId</param>
private string GoCodeUrl(string shareOpenId)
{
    string url = System.Web.HttpContext.Current.Request.Url.AbsoluteUri +
"&s=" + shareOpenId; //添加分享者 OpenId
    return OAuth.GetAuthorizeUrl(appID, url, "STATE", OAuthScope.snsapi_base);
}
/// <summary>
/// 网页授权接口第二步
/// 解析 Code 并获取当前访问者真正的 OpenId
/// </summary>
/// <param name="parameters">URL 参数</param>
/// <returns>真正的 OpenId</returns>
private OAuthAccessTokenResult GetRealOpenId(string code)
{
    OAuthAccessTokenResult result = new OAuthAccessTokenResult();
    try
    {
        result = OAuth.GetAccessToken(appID, appsecret, code);
    }
    catch (Exception)
    {
    }
    return result;
}
```

7.5 阅读、分享统计实现

在了解了如何获取访问来源、识别访问者与识别分享者后，接下来着手实现对阅读、分享与来源的统计。

首先打开 Microsoft Visual Studio 2012 开发环境，建立一个名为 Sample_4 的 ASP.NET 空 Web 应用程序，参照第 4 章的 Sample_2 项目建立好微信公众号访问接口。然后在 CustomMessageHandler.cs 中，按照 7.4.2 节所述的分享者识别方法，重写文本消息处理方法 OnTextRequest，在回复的图文消息的链接中，加入发送文本消息的微信个人用户的 OpenId。

```
    /// <summary>
    /// 处理文字请求
    /// </summary>
    /// <returns></returns>
    protected override IResponseMessageBase OnTextRequest(RequestMessageText
requestMessage)
    {
        var responseMessage = CreateResponseMessage<ResponseMessageNews>();
        responseMessage.Articles.Add(new Article()
        {
            Title = "首页",
            Description = "单击进入首页",
            PicUrl = "",
            Url = System.Configuration.ConfigurationManager.AppSettings["site"] +
"/WeixinPageIndex.aspx?u=" + requestMessage.FromUserName
        });
        return responseMessage;
    }
```

7.5.1 内存数据库实现数据存取

由于篇幅所限，阅读、分享统计的示例程序使用内存数据库保存阅读和统计记录。在实际项目中，需要按第 6 章讲解的三层架构模式设计阅读、统计记录的数据库、数据访问层及业务逻辑层。

数据实体 PageNavEntity 用于保存阅读记录，在阅读记录中包含微信个人用户访问的页面地址、访问来源、访问者、分享者等信息。

```
/// <summary>
/// 页面访问记录
/// </summary>
public class PageNavEntity
{
    /// <summary>
    /// Id
    /// </summary>
    public string Id { get; set; }
    /// <summary>
    /// 页面地址
    /// </summary>
    public string Url { get; set; }
    /// <summary>
    /// 访问来源
    /// </summary>
    public NavFrom From { get; set; }
    /// <summary>
    /// 访问者微信 OpenId
    /// </summary>
    public string NavOpenId { get; set; }
    /// <summary>
    /// 当访问来源为朋友圈时的分享者微信 OpenId
    /// </summary>
    public string ShareOpenId { get; set; }
    /// <summary>
    /// 访问时间
    /// </summary>
    public DateTime VisitTime { get; set; }
}
```

数据实体 PageShareEntity 用于保存分享记录，分享记录中包含微信个人用户分享的页面地址、由枚举类型 ShareType 表示的分享类型、分享者、上一级分享者等信息。

```
/// <summary>
/// 页面分享记录
/// </summary>
public class PageShareEntity
{
    /// <summary>
    /// Id
```

```
        /// </summary>
        public string Id { get; set; }
        /// <summary>
        /// 页面地址
        /// </summary>
        public string Url { get; set; }
        /// <summary>
        /// 分享类型
        /// </summary>
        public ShareType From { get; set; }
        /// <summary>
        /// 分享者微信 OpenId
        /// </summary>
        public string ShareOpenId { get; set; }
        /// <summary>
        /// 上一级分享者微信 OpenId
        /// </summary>
        public string ParentShareOpenId { get; set; }
        /// <summary>
        /// 分享时间
        /// </summary>
        public DateTime ShareTime { get; set; }
    }
    /// <summary>
    /// 分享类型
    /// </summary>
    public enum ShareType
    {
        /// <summary>
        /// 分享给好友
        /// </summary>
        Friend,
        /// <summary>
        /// 分享到朋友圈
        /// </summary>
        Timeline,
        /// <summary>
        /// 未知
        /// </summary>
        Unknown
    }
```

还需要一个内存数据库实体 Statistics，用于保存所有的阅读、分享记录。

```
/// <summary>
/// 访问与分享记录
/// </summary>
public class Statistics
{
    /// <summary>
    /// 页面访问记录
    /// </summary>
    public List<PageNavEntity> PageNav { get; set; }
    /// <summary>
    /// 页面分享记录
    /// </summary>
    public List<PageShareEntity> PageShare { get; set; }
}
```

内存数据库的操作类为 StatisticsBll，通过其中的静态构造函数，初始化一个全局唯一的内存数据库 statistics。StatisticsBll 类中提供了一些存取数据的基本方法。

```
public class StatisticsBll
{
    /// <summary>
    /// 访问与分享记录的内存数据库
    /// </summary>
    private static Statistics _statistics;
    /// <summary>
    /// 静态构造函数
    /// </summary>
    static StatisticsBll()
    {
        //初始化内存数据库
        _statistics = new Statistics();
        _statistics.PageNav = new List<PageNavEntity>();
        _statistics.PageShare = new List<PageShareEntity>();
    }
    /// <summary>
    /// 添加访问记录
    /// </summary>
    /// <param name="entity"></param>
    /// <returns></returns>
```

```
        public static void InsertPageNav(PageNavEntity entity)
        {
            entity.VisitTime = DateTime.Now;
            _statistics.PageNav.Add(entity);
        }
        /// <summary>
        /// 添加分享记录
        /// </summary>
        /// <param name="entity"></param>
        /// <returns></returns>
        public static void InsertPageShare(PageShareEntity entity)
        {
            entity.ShareTime = DateTime.Now;
            _statistics.PageShare.Add(entity);
        }
        /// <summary>
        /// 获取访问记录
        /// </summary>
        /// <returns></returns>
        public static List<PageNavEntity> GetPageNavList()
        {
            return _statistics.PageNav;
        }
        /// <summary>
        /// 获取分享记录
        /// </summary>
        /// <returns></returns>
        public static List<PageShareEntity> GetPageShareList()
        {
            return _statistics.PageShare;
        }
    }
```

7.5.2 实现阅读、分享数据记录

在 Sample_4 项目中，建立一个 Web 窗体 WeixinPageIndex.aspx。接下来以 WeixinPageIndex 页面为例，按照前几节所介绍的方法实现阅读、分享记录。在实际项目中，可将本节的实现方式抽象出一个基类，微信公众号的所有页面均继承于该基类，即可实现所有页面的阅读、分享数据记录。

每次页面访问都会产生一条阅读记录，阅读信息的获取与保存，直接在 WeixinPageIndex 的页面加载方法 Page_Load 中实现。调用 7.3 节中给出的获取访问来源的方法 GetNavFromType，以及 7.4 节中给出的获取访问者及分享者的方法 GetNavOpenId 即可获取阅读信息。

获取访问者 OpenId 需要调用微信公众平台的“网页授权接口”，需要将微信公众号的开发者凭据包含在页面中。

```
    public partial class WeixinPageIndex : System.Web.UI.Page
    {
        /// <summary>
        /// 从微信公众平台获取的开发者凭据
        /// </summary>
        const string appID = "wx637267c8ab5abf7f";
        /// <summary>
        /// 从微信公众平台获取的开发者凭据
        /// </summary>
        const string appsecret = "0703c5e946a658409b48386e45295e4a";
        protected void Page_Load(object sender, EventArgs e)
        {
            NameValueCollection parameters = System.Web.HttpContext.Current.
Request.Params;
            //取得链接中的分享者 OpenId
            string shareOpenId = parameters["s"];
            //获取访问者的 OpenId
            string navOpenId = GetNavOpenId();
            if (navOpenId != null)
            {
                NavStatistics(navOpenId, shareOpenId);
                //传递给页面的访问者 OpenId
                ViewState["navOpenId"] = navOpenId;
                //传递给页面的分享者 OpenId
                ViewState["shareOpenId"] = shareOpenId;
            }
        }
        /// <summary>
        /// 记录页面访问
        /// </summary>
        /// <param name="navOpenId">访问者微信 openid</param>
        /// <param name="shareOpenId">当访问来源为朋友圈时的分享者微信 openid</
param>
```

```
        private void NavStatistics(string navOpenId, string shareOpenId)
        {
            //获取访问来源
            NavFrom fromType = GetNavFromType();
            //构造访问记录
            var pageNav = new PageNavEntity()
            {
                Id = Guid.NewGuid().ToString(),
                Url = GetOrigenalUrl(),
                NavOpenId = navOpenId,
                ShareOpenId = shareOpenId,
                From = fromType,
                VisitTime = DateTime.Now
            };
            //访问记录写入数据库
            StatisticsBll.InsertPageNav(pageNav);
        }
    }
```

分享数据的获取，通过在页面中引用 7.2 节中给出的捕获发送给朋友、分享到朋友圈等动作的 StatisticsWeChatAPI.js 实现。获取到的分享数据，通过 dataForWeixin.friendCirclecallback 及 dataForWeixin.friendcallback 回调函数，传递给后台保存到数据库中。 WeixinPageIndex 的页面代码如下：

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WeixinPageIndex.
aspx.cs" Inherits="Sample_4.WeixinPageIndex" %>
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
       <title></title>
    </head>
    <body>
       <%-- 所有的跳转页面，加上访问者与分享者的 OpenId --%>
       <a href="WeixinPageSubPage.aspx?u=<%= ViewState["navOpenId"] as
string %>&s=<%= ViewState["shareOpenId"] as string %>">WeixinPageSubPage</a>
       <form id="form1" runat="server">
       <div>
       </div>
       </form>
    </body>
```

```
    </html>
    <script>
        (function () {
            var url = location.href;
            dataForWeixin.appId = "";
            dataForWeixin.MsgImg = "";
            dataForWeixin.TLImg = "";
            dataForWeixin.url = url;
            dataForWeixin.title = 'Title';
            dataForWeixin.desc = 'Description';
            dataForWeixin.navOpenId = '<%= ViewState["navOpenId"] as string %>';
            dataForWeixin.shareOpenId = '<%= ViewState["shareOpenId"] as string %>';
            //转发到朋友圈的回调函数，向后台传递转发记录
            dataForWeixin.friendCirclecallback = function () {
                //AJAX 请求
                $.ajax({
                    type: "get",
                    url: 'Share.aspx?type=timeline&url=' + encodeURIComponent
(dataForWeixin.url) + "&u=" + dataForWeixin.navOpenId + "&s=" + dataForWeixin.
shareOpenId,
                    beforeSend: function () {
                    },
                    success: function () {
                    },
                    complete: function () {
                    },
                    error: function () {
                    }
                });
            };
            //转发给朋友的回调函数，向后台传递转发记录
            dataForWeixin.friendcallback = function (res) {
                //AJAX 请求
                $.ajax({
                    type: "get",
                    url: 'Share.aspx?type=friend&url=' + encodeURIComponent
(dataForWeixin.url) + "&navOpenId=" + dataForWeixin.navOpenId +
"&shareOpenId=" + dataForWeixin.shareOpenId,
                    beforeSend: function () {
                    },
                    success: function () {
```

```
            },
            complete: function () {
            },
            error: function () {
            }
        });
    };
})();
</script>
```

为了在后台保存分享记录，还需要建立一个 Web 窗体 Share.aspx，接收 dataForWeixin.friendCirclecallback 及 dataForWeixin.friendcallback 回调函数传递的分享数据。在 Share 的页面加载方法 Page_Load 中，实现识别分享记录类型，构造分享记录数据实体，并保存到数据库的功能。

```
/// <summary>
/// 接收分享记录信息并保存到数据库
/// </summary>
public partial class Share : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string typeStr = Request.QueryString["type"];
        if (!string.IsNullOrEmpty(typeStr))
        {
            //识别分享类型
            ShareType type = ShareType.Unknown;
            switch (typeStr)
            {
                case "timeline":
                    type = ShareType.Timeline;
                    break;
                case "friend":
                    type = ShareType.Friend;
                    break;
            }
            //构造分享记录
            var pageShare = new PageShareEntity()
            {
                Id = Guid.NewGuid().ToString(),
```

```
                Url = GetOrigenalUrl(System.Web.HttpContext.Current.Request.
QueryString["url"]),
                ParentShareOpenId = System.Web.HttpContext.Current.Request.
QueryString["s"],
              ShareOpenId = System.Web.HttpContext.Current.Request. QueryString
["u"],
                From = type,
                ShareTime = DateTime.Now
              };
              //保存分享记录
              StatisticsBll.InsertPageShare(pageShare);
          }
      }
      /// <summary>
      /// 获取不含统计相关参数的页面地址
      /// </summary>
      /// <param name="url">网址</param>
      /// <returns>不含统计相关参数的页面地址</returns>
      private string GetOrigenalUrl(string url)
      {
          url = System.Web.HttpUtility.UrlDecode(url);
          Uri uri = new Uri(url);
          StringBuilder urlBuilder = new StringBuilder();
          //获取不含 QueryString 的 URL
          urlBuilder.Append("http://")
              .Append(uri.Host)
              .Append(uri.AbsolutePath)
              .Append("?");
          //构造移除统计相关参数的 Query
          Dictionary<string, string> queryString = uri.Query.Replace("?",
"").Split('&').Where(p => !string.IsNullOrEmpty(p)).ToDictionary(p => p.Split
('=')[0], p => p.Split('=')[1].Split('#')[0]);
          foreach (var key in queryString.Keys)
          {
              if (key != "s" && key != "u" && key != "from" && key != "code"
&& key != "state")
              {

urlBuilder.Append(key).Append("=").Append(queryString[key]).Append("&");
              }
          }
```

```
            return urlBuilder.ToString();
        }
    }
```

7.5.3 实现阅读、分享统计

获取到的阅读与分享记录，需要显示出来才能进行分析。为了更深入地分析微信公众号的运营情况，还需要对阅读与分享记录按时间进行统计，然后以曲线图的形式展示出来。

统计曲线图的绘制使用 HighCharts 实现。HighCharts 是一个功能强大、开源、美观、图表丰富、兼容绝大多数浏览器的纯 js 图表库。在 Highcharts 中文网（http://www.hcharts.cn/）中有对 HighCharts 控件的详细教程，感兴趣的读者可以访问该网站深入了解 HighCharts 的功能及使用方法。

为了更方便地进行统计曲线图的绘制，我们将使用 HighCharts 进行绘图的相关 JavaScript 代码封装在 publiclinecharts.js 文件中。

```
var highcharts = {
    opts: {
        getStatisticsUrl: "", //获取图表数据的地址
        titletext: "", //标题
        ytext: "", //y轴显示文本
        startyear: 0, //开始年份
        startmonth: 0, //开始月份
        startday: 0, //开始天数
        lineinterval: 0, //以多少天为间隔显示一条竖线（毫秒）
        pointInterval: 0, //显示图表数据点的间隔（毫秒）
        countArray: [], //图表的数据
        formid: "", //需要填充的DIV的ID
        seriesname: "", //显示数据列的名称
        unit: ""
    },
    displayMode: 0,
    init: function (opts) {
        highcharts.getStatistics(opts);
    },
    //获取图的数据
    getStatistics: function (opts) {
        jQuery.ajax({
```

```
            url: opts.getStatisticsUrl,
            type: "get",
            dataType: "json",
            data: { plat: opts.plat },
            success: function (repJson) {
               if (repJson) {
                  highcharts.extFunction.PreDrawMethod(repJson);
                  opts.countArray.push(highcharts.getSeries(opts,
opts.seriesname, repJson.Statistics));
                  highcharts.drawChart(opts);
               }
            },
            complete: function (xhr, ts) {
               xhr = null;
            }
         });
      },
      //画图
      drawChart: function (opts) {
         var chart = new Highcharts.Chart({
            chart: {
               renderTo: opts.formid,
               type: 'line',
               marginRight: 30,
               marginBottom: 25
            },
            title: {
               text: opts.titletext,
               align: 'left'
            },
            xAxis: {
               type: 'datetime',
               labels: {
                  formatter: function () {
                     return highcharts.getFormatDate2(this.value);
                  },
                  style: {
                     color: '#2f7ed8'
                  }
               },
               gridLineWidth: 1,
```

```
                tickInterval: opts.lineinterval// one week
            },
            yAxis: {
                title: {
                    text: opts.ytext
                },
                plotLines: [{
                    value: 0,
                    width: 1,
                    color: '#808080'
                }],
                min: 0
            },
            tooltip: {
                formatter: function () {
                        return highcharts.getFormatDate2(this.x) + ': <span
style="color:red">' + this.y + '<span>' + opts.unit;
                }
            },
            credits: {
                enabled: false//不显示 HighCharts 版权信息
            },
            legend: {
                enabled: true,
                align: 'right',
                verticalAlign: 'top',
                floating: true,
                y: 0,
                borderWidth: 0,
                margin: 0,
                symbolPadding: 5,
                symbolWidth: 12,
                itemStyle: {
                    color: '#636363'
                }
            },
            series: opts.countArray
        });
    },
    getSeries: function (opts, seriesname, statistics) {
        var data = {
```

```
            name: seriesname,
            data: statistics,
            pointStart: Date.UTC(opts.startyear, opts.startmonth - 1, opts.
startday),
            pointInterval: opts.pointInterval
        };
        return data;
    },
    ///格式化时间
    getFormatDate2: function (v) {
        var d = new Date(v);
        d.setHours(d.getHours() - 8);
        return d.getFullYear() + "-" + (d.getMonth() + 1) + "-" + d.getDate()
+ " " + d.getHours() + "点";
    },
    extFunction: {
        //画图前预处理方法
        PreDrawMethod: function (repJson) {
        }
    }
}
```

统计曲线图绘制所需的数据，由 Data.aspx 提供。

```
/// <summary>
/// 为页面 HighCharts 画图控件提供数据
/// </summary>
public partial class Data : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        string result = "";
        string typeStr = System.Web.HttpContext.Current.Request.QueryString
["type"];
        if (!string.IsNullOrEmpty(typeStr))
        {
            switch (typeStr)
            {
                case "navChart": //页面访问图
                result = JsonConvert.SerializeObject(GetPageNavStatistics());
                    break;
                case "shareChart": //页面分享图
```

```
                    result = JsonConvert.SerializeObject
(GetPageShareStatistics());
                    break;
                }
            }
            //将 HighCharts 绘图所需的数据返回给页面
            HttpResponse response = System.Web.HttpContext.Current.Response;
            response.ContentType = "application/json";
            response.Write(result);
            response.End();
        }
        /// <summary>
        /// 获取页面访问统计信息
        /// </summary>
        /// <returns></returns>
        private ChartData GetPageNavStatistics()
        {
            //取过去两天的数据进行统计
            DateTime startTime = DateTime.Now.AddDays(-3);
            DateTime endTime = DateTime.Now.AddDays(1);
            List<PageNavEntity> temp = StatisticsBll.GetPageNavList();
            List<decimal> statistics = new List<decimal>();
            //HighCharts 时间轴的起始时间
            ChartData chartData = new ChartData
            {
                StartYear = startTime.Year,
                StartDay = startTime.Day,
                StartMonth = startTime.Month
            };
            //生成按小时统计的数据
            while (startTime < endTime)
            {
                statistics.Add(temp.FindAll(e => e.VisitTime >= startTime &&
e.VisitTime < startTime.AddHours(1)).Count());
                startTime = startTime.AddHours(1);
            }
            chartData.Statistics = statistics.ToArray();
            return chartData;
        }
        /// <summary>
        /// 获取页面分享统计信息
```

```
        /// </summary>
        /// <returns></returns>
        private ChartData GetPageShareStatistics()
        {
            //取过去两天的数据进行统计
            DateTime startTime = DateTime.Now.AddDays(-3);
            DateTime endTime = DateTime.Now.AddDays(1);
            List<PageShareEntity> temp = StatisticsBll.GetPageShareList();
            List<decimal> statistics = new List<decimal>();
            //HighCharts 时间轴的起始时间
            ChartData chartData = new ChartData
            {
                StartYear = startTime.Year,
                StartDay = startTime.Day,
                StartMonth = startTime.Month
            };
            //生成按小时统计的数据
            while (startTime < endTime)
            {
                statistics.Add(temp.FindAll(e => e.ShareTime >= startTime &&
e.ShareTime < startTime.AddHours(1)).Count());
                startTime = startTime.AddHours(1);
            }
            chartData.Statistics = statistics.ToArray();
            return chartData;
        }
    }
```

最后，在 Sample_4 项目中建立一个 Web 窗体 Statistics.aspx，该页面负责以列表的形式呈现所有的阅读记录，以曲线图的形式显示最近几天每小时的阅读统计数据。

Statistics.aspx 的前端代码如下：

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Statistics.aspx.cs"
Inherits="Sample_4.Statistics1" %>
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>统计</title>
        <%-- Bootstrap --%>
```

```
        <link href="Css/bootstrap.min.css" rel="stylesheet" type="text/css" />
        <script src="Scripts/bootstrap.min.js" type="text/javascript"></script>
        <script  src="Scripts/jquery-1.9.1.min.js"  type="text/javascript">
</script>
        <%-- HighCharts 用于图表显示 --%>
        <link href="Css/highcharts/charts.css" rel="stylesheet" type="text/
css" />
        <script src="Scripts/HighCharts/highcharts.js" type="text/javascript
"></script>
        <script src="Scripts/HighCharts/highcharts-more.js" type="text/
javascript" ></script>
        <script src="Scripts/HighCharts/publiclinecharts.js" type="text/
javascript"></script>
    </head>
    <body>
        <div class="row-fluid">
            <div class="span12">
                <h3>访问记录</h3>
                <%-- 访问记录统计图 --%>
                <div class="box">
                    <div class="box-content">
                        <div class="row" style="margin-top: 30px; margin-right:
15px;">
                            <div class="area">
                                <div id="page-nav-chart">
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
                <%-- 访问记录列表 --%>
                <div class="maincontentinner1" style="margin-left: 20px">
                      <div id="Div12" class="dataTables_wrapper">
                          <table   id="page-nav-table"   class="table   table-
bordered responsive dataTable">
                              <%-- 访问记录列表列名 --%>
                              <thead>
                                  <tr>
                                      <th>
                                          页面地址
                                      </th>
```

```
                                <th>
                                    访问来源
                                </th>
                                <th>
                                    访问者 openid
                                </th>
                                <th>
                                    分享者 openid
                                </th>
                                <th>
                                    访问时间
                                </th>
                            </tr>
                        </thead>
                        <tbody id="page-nav-table-body">
                            <%-- 一行一行生成访问记录列表 --%>
                            <% foreach (Sample_4.PageNavEntity entity in
(ViewState["NavList"] as List<Sample_4.PageNavEntity>))
                                { %>
                                <tr class="gradeX odd">
                                    <td>
                                        <%= entity.Url%>
                                    </td>
                                    <td class=" ">
                                        <%= entity.From.ToString()%>
                                    </td>
                                    <td class=" ">
                                        <%= entity.NavOpenId%>
                                    </td>
                                    <td class=" ">
                                        <%= entity.ShareOpenId%>
                                    </td>
                                    <td class=" ">
                                        <%= entity.VisitTime.ToString()%>
                                    </td>
                                </tr>
                            <% } %>
                        </tbody>
                    </table>
                </div>
            </div>
```

```
        </div>
    </div>
    <script>
        //图表参数
        var pageNavChartOpts={
            getStatisticsUrl:'Data.aspx?type=navChart', //读取数据的访问地址
            titletext:"",
            ytext:"",
            startyear: 0,
            startmonth:0,
            startday:0,
            lineinterval: 3600 * 1000, //竖线以1小时为间隔显示
            pointInterval: 3600 * 1000,//点以1小时为间隔显示
            countArray:[],
            formid:"page-nav-chart", //图表容器ID
            seriesname:"访问次数",
            unit:"次"
        };
        jQuery(function () {
            //使用HighCharts绘制图表
            highcharts.extFunction.PreDrawMethod = function (repJson) {
                pageNavChartOpts.startyear = repJson.StartYear;
                pageNavChartOpts.startmonth = repJson.StartMonth;
                pageNavChartOpts.startday = repJson.StartDay;
                highcharts.displayMode = repJson.DisplayMode;
                pageNavChartOpts.lineinterval = repJson.LineInterval;
                pageNavChartOpts.pointInterval = repJson.PointInterval;
            };
            highcharts.init(pageNavChartOpts);
        });
    </script>
</body>
</html>
```

Statistics.aspx 的后端代码如下：

```
public partial class Statistics1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        //传递给页面显示的记录列表
        ViewState["NavList"] = StatisticsBll.GetPageNavList();
```

```
        ViewState["ShareList"] = StatisticsBll.GetPageShareList();
    }
}
```

Sample_4 项目的完整项目结构如图 7-1 所示。

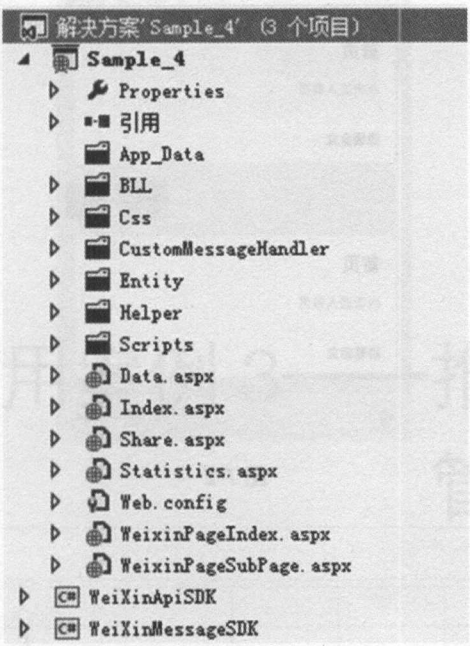


图 7-1

7.6　部署及测试体验

在 Sample_4 项目本地调试通过后，需要将项目上传到 AppHarbor 中进行部署。让微信服务器能访问到我们的程序，进行实际测试。AppHarbor 部署的详细步骤在第 4 章有完整讲解。

使用微信客户端关注微信公众号。向该微信公众号发送“首页”，单击回复的图文，进入 WeixinPageIndex.aspx 页面，如图 7-2 所示。

然后在电脑的浏览器中打开 Statistics.aspx 页面，将看到如图 7-3 所示的统计曲线图与详细记录列表。

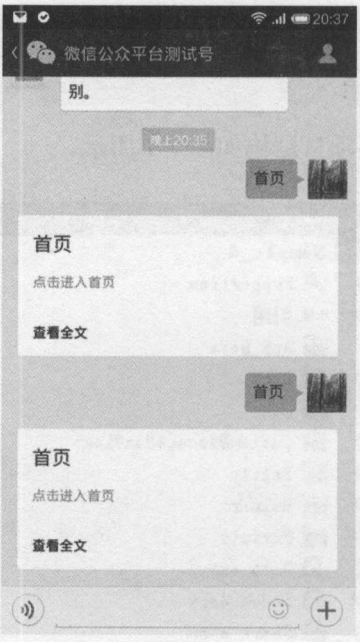
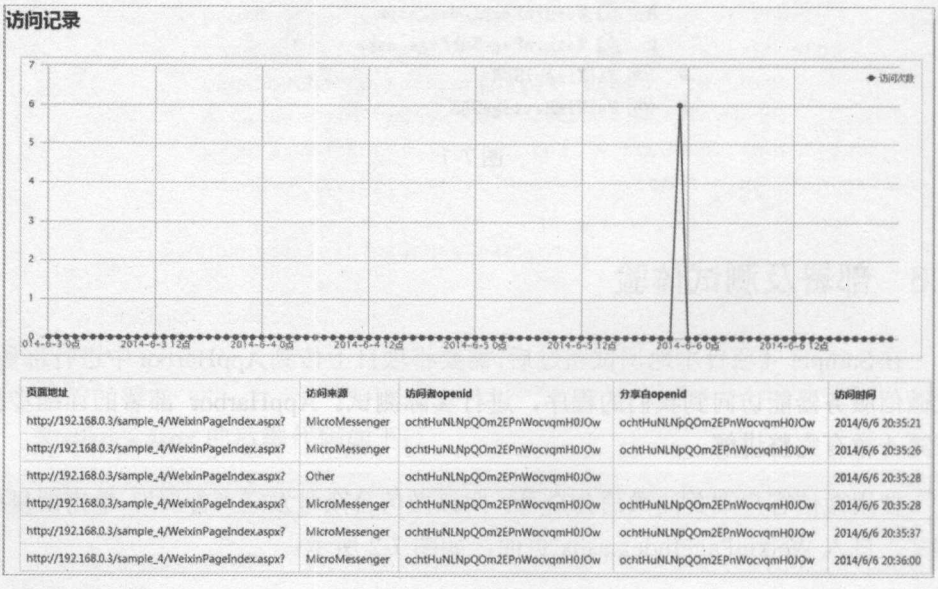


图 7-2



页面地址	访问来源	访问者openid	分享自openid	访问时间
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	MicroMessenger	ochtHuNLNpQOm2EPnWocvqmH0JOw	ochtHuNLNpQOm2EPnWocvqmH0JOw	2014/6/6 20:35:21
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	MicroMessenger	ochtHuNLNpQOm2EPnWocvqmH0JOw	ochtHuNLNpQOm2EPnWocvqmH0JOw	2014/6/6 20:35:26
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	Other	ochtHuNLNpQOm2EPnWocvqmH0JOw		2014/6/6 20:35:28
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	MicroMessenger	ochtHuNLNpQOm2EPnWocvqmH0JOw	ochtHuNLNpQOm2EPnWocvqmH0JOw	2014/6/6 20:35:28
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	MicroMessenger	ochtHuNLNpQOm2EPnWocvqmH0JOw	ochtHuNLNpQOm2EPnWocvqmH0JOw	2014/6/6 20:35:37
http://192.168.0.3/sample_4/WeixinPageIndex.aspx?	MicroMessenger	ochtHuNLNpQOm2EPnWocvqmH0JOw	ochtHuNLNpQOm2EPnWocvqmH0JOw	2014/6/6 20:36:00

图 7-3

第 8 章
商用案例 3——推广渠道管理系统

本章主要讲解如何使用微信公众平台提供的“生成带参数二维码接口”与“用户管理接口”，实现生成能标记不同推广渠道的微信公众号二维码，记录分配给不同推广渠道的二维码被扫描的信息。进而在 Microsoft Visual Studio 2012 开发环境中，使用 C#语言开发出对不同推广渠道进行管理，并统计和分析不同推广渠道推广效果的推广渠道管理系统。

8.1 微信公众号推广综述

当微信公众号的功能搭建完毕，正式运营一段时间后，几乎所有的微信公众号都会面临一个问题，关注微信公众号的用户太少，如何增加微信公众号的关注

人数。

与微博的开放、弱关系社交属性相反，微信是一个强关系、密友属性的社交工具。为了保持微信良好的用户体验，微信团队对微信个人用户的隐私保护做了很多努力和工作，所有可能对微信个人用户造成打扰的功能，微信团队都做了严格的限制甚至屏蔽。这导致了微信公众号在微信中几乎没有办法获取更多用户的关注，微信公众号无法查找未关注自己的微信个人用户信息，也无法给未关注自己的微信个人用户发送消息。甚至连让未关注的微信个人用户看到微信公众号的唯一渠道微信朋友圈，微信团队都做了严格的规定，禁止微信公众号在朋友圈中进行推广。最近几个月最火的微信公众号推广方式“微信朋友圈集赞”，刚刚被微信团队发布公告明令禁止，所有在发布公告后使用“微信朋友圈集赞”进行推广的卫星公众号，都将被微信团队封停账号。

微信团队开放微信公众平台的初衷，以及一直以来对微信公众平台的定位，都是使用微信公众号来更好地为用户提供服务。微信公众平台是一个提供用户服务的平台而不是进行营销的平台。所有在微信内进行营销推广的方式，对微信团队来说，都是在破坏微信的用户体验，微信团队必将全力限制、禁止及处罚。在微信内进行营销推广，只是利用微信公众平台规则的漏洞和不完善，只能取得短期效果，并且存在随时被微信团队封号的巨大风险。

要想长期、持续、稳定的增加微信公众号的关注人数，只能使用微信公众号的二维码，在微信之外进行推广，利用线上、线下的正规、成熟的推广渠道进行推广。使用户通过扫描二维码关注微信公众号，是微信团队认可和推荐的推广方式。当然，要想取得好的推广效果，还需要不断完善微信公众号的功能，以及专业的、有创意的推广策划方案。

根据笔者的经验，微信公众号二维码的推广渠道主要有以下几种：

1. 在自有实体场所中摆放微信公众号的二维码。

2. 在合作商户的实体场所中摆放微信公众号的二维码。

3. 在官方网站中显示微信公众号的二维码。

4. 在商品包装上印制微信公众号的二维码。

5. 在快递包裹中放入微信公众号的二维码。

6. 在线下的平面媒体广告、户外广告牌、电视广告、楼宇广告等传统媒体广

告中显示微信公众号的二维码。

7. 在线上的百度网盟、网页广告栏、视频等广告中显示微信公众号的二维码。

8. 在会议、培训等活动现场摆放微信公众号的二维码。

9. 在在线客服聊天工具中发送微信公众号的二维码。

如果想让更多的已有客户关注微信公众号，那么可以在自有经营场所或产品上展示微信公众号的二维码。餐厅可以在餐桌上摆放二维码，商场门店可以在收银台前摆放二维码，制造商可以在商品包装上印制二维码，电商企业可以在快递的包裹中放入二维码，互联网企业可以在官方网站中显示二维码。

如果想吸引更多的潜在客户关注微信公众号，则可以在原本的线上、线下广告中加入微信公众号的二维码。

单独一个微信公众号的二维码很难让用户有扫描二维码关注的冲动。所有的二维码旁都应该辅以“关注有礼”、“关注享受优惠”等营销活动说明。

8.2　推广渠道管理系统功能及设计

8.2.1　推广渠道管理系统需求

在微信公众号二维码推广中，营销活动策划的好坏和投放渠道的匹配度，将直接影响最终关注该微信公众号的用户数量，决定最终的推广效果。如果营销活动的策划不能引起用户的关注冲动，广告投放渠道选择不合适，那么将导致企业虽然投入很多精力和资金，但是只有寥寥数人关注微信公众号，微信公众号推广的投入产出不成正比。

在一个微信公众号刚开始推广时，市场人员很难从众多的投放渠道中找到推广效果最好的组合方案，营销策划人员也很难第一次就给出能很好吸引用户关注的策划案。只有经过一段时间的推广尝试，监测各种推广方式的效果，不断对过往推广效果数据进行统计、分析、总结，才能逐渐找到一套符合该微信公众号实际情况的，行之有效的推广策略，从而最终达到预定的市场推广目标。这正是推广渠道管理系统要解决的问题。

8.2.2 推广渠道管理系统功能

要了解各种推广渠道的二维码推广效果，首先需要对不同的推广渠道分配不同的微信公众号二维码。微信个人用户使用微信扫描不同渠道的微信公众号二维码都能关注微信公众号。

推广渠道管理系统在用户关注微信公众号时，需要能识别该次关注是从哪个推广渠道产生的，并将关注用户的微信 OpenId、推广渠道等信息作为一条推广效果记录保存起来。

微信公众平台的“生成带参数二维码接口”能够生成含不同数字标识的微信公众号二维码，推广渠道管理系统可以利用该接口为不同的推广渠道分配含有各不重复数字标识的微信公众号二维码。

当微信个人用户使用微信扫描了一个带参数的二维码，在微信服务器发给微信公众号服务器的关注事件或扫描事件中，将包含二维码的参数。推广渠道管理系统通过识别事件中的二维码参数，即可记录每个推广渠道带来的二维码扫描记录。

由于二维码扫描记录中，仅记录了扫描用户的微信 OpenId，还需要通过微信公众平台提供的“用户管理接口”获取该用户的微信基本信息，才能形成一条完整的推广渠道效果记录。

对每个推广渠道或一系列推广渠道组合的推广渠道效果记录进行统计，对比每个推广渠道或渠道组合带来的关注人数，可以找出最有效的推广渠道组合。对推广渠道带来的关注人群特征进行分析，可以指导后续营销活动策划。

推广渠道管理系统应包含以下几个基本功能：

1. 获取微信公众号关注者的基本信息。
2. 对推广渠道进行分组。
3. 对推广渠道进行基本设置。
4. 获取每个推广渠道的二维码。
5. 对每次扫描进行记录。
6. 显示包含扫描者微信基本信息的扫描记录，供推广效果评估。

8.2.3　数据表设计

根据 8.2.1 节的分析，一个最基本的推广渠道管理系统数据库需要包含以下 4 张表：

1. 微信用户信息表 WeixinUserInfo
2. 推广渠道类型（渠道分组）表 ChannelType
3. 推广渠道表 Channel
4. 推广渠道二维码扫描记录表 ChannelScan

生成数据库的 SQL 语句如下：

```
-- ------------------------------------------------
-- Creating all tables
-- ------------------------------------------------
-- Creating table 'Channel'
-- 推广渠道表
CREATE TABLE [Channel] (
    --主键
   [ID] int IDENTITY(1,1) NOT NULL,
    --生成带参数二维码接口的场景值 ID
    --扫描二维码产生的事件中将包含该场景值 ID
    --以此识别不同推广渠道产生的扫描
   [SceneId] int  NOT NULL,
    --所属渠道类型 ID
   [ChannelTypeId] int  NOT NULL,
    --渠道名称
   [Name] nvarchar(4000)  NOT NULL,
    --分配给渠道的含场景值 ID 的二维码图片名称
   [Qrcode] ntext  NOT NULL
);
GO
-- Creating table 'ChannelType'
-- 推广渠道类型（渠道分组）表
CREATE TABLE [ChannelType] (
    --主键
   [ID] int IDENTITY(1,1) NOT NULL,
    --渠道类型名称
   [Name] nvarchar(4000)  NOT NULL
);
```

```
GO
-- Creating table 'ChannelScan'
-- 推广渠道二维码扫描记录表
CREATE TABLE [ChannelScan] (
    --主键
    [ID] int IDENTITY(1,1) NOT NULL,
    --扫描二维码的微信个人用户 OpenId
    [OpenId] nvarchar(4000)  NOT NULL,
    --扫描类型
    [ScanType] smallint  NOT NULL,
    --所属渠道
    [ChannelId] int  NOT NULL,
    --扫描时间
    [ScanTime] datetime  NOT NULL
);
GO
-- Creating table 'WeixinUserInfo'
-- 微信用户信息表
CREATE TABLE [WeixinUserInfo] (
    --微信用户的 OpenId
    [OpenId] nvarchar(4000)  NOT NULL,
    --昵称
    [NickName] nvarchar(4000)  NOT NULL,
    --用户的头像链接
    [HeadImgUrl] nvarchar(4000)  NOT NULL,
    --用户的语言，简体中文为 zh_CN
    [Language] nvarchar(4000)  NOT NULL,
    --性别
    [Sex] smallint  NOT NULL,
    --用户所在城市
    [City] nvarchar(4000)  NOT NULL,
    --用户所在省份
    [Province] nvarchar(4000)  NOT NULL,
    --用户所在国家
    [Country] nvarchar(4000)  NOT NULL,
    --用户关注时间
    [Subscribe_time] bigint  NOT NULL,
    --主键
    [ID] int IDENTITY(1,1) NOT NULL
);
GO
```

```
-- --------------------------------------------------
-- Creating all PRIMARY KEY constraints
-- --------------------------------------------------
-- Creating primary key on [ID] in table 'Channel'
ALTER TABLE [Channel]
ADD CONSTRAINT [PK_Channel]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'ChannelType'
ALTER TABLE [ChannelType]
ADD CONSTRAINT [PK_ChannelType]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'ChannelScan'
ALTER TABLE [ChannelScan]
ADD CONSTRAINT [PK_ChannelScan]
   PRIMARY KEY ([ID] );
GO
-- Creating primary key on [ID] in table 'WeixinUserInfo'
ALTER TABLE [WeixinUserInfo]
ADD CONSTRAINT [PK_WeixinUserInfo]
   PRIMARY KEY ([ID] );
GO
```

8.3　推广渠道管理系统实现

下面，我们采用第 6 章讲述的三层架构实现推广渠道管理系统。

首先打开 Microsoft Visual Studio 2012 开发环境，建立一个名为 Sample_5 的 ASP.NET 空 Web 应用程序。然后在项目中引用 WeiXinMessageSDK，建立 Index.aspx 用于接收微信服务器请求。

接下来在 Sample_5 项目中建立好用于存放各层代码的文件夹 ORM、DAL、ViewEntities、BLL 等。

Sample_5 项目的项目结构如图 8-1 所示。

数据访问框架（ORM）按照 6.3.2 节所讲述的方法建立，在此不再敷述。建立好的实体数据模型名称为：Channel.edmx。

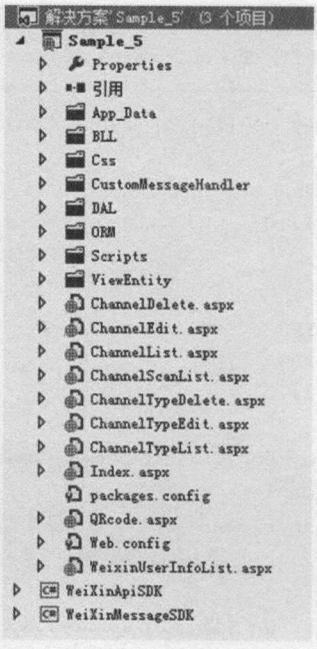


图 8-1

8.3.1 实现数据访问层

在建立好 ORM 后，根据 ORM 建立 DLL 层。

6.3.4 节中已经建立过 EntityFramework 的连接方法 DbContextFactory 及 DLL 层的基类 BaseDll。将 Sample_3 项目中的 DbContextFactory.cs、BaseDll.cs 及 EntityObjectExt.cs 复制到 Sample_5 项目中，修改 DbContextFactory.cs 文件中的上下文名称为“ChannelContent”，修改后的 DbContextFactory.cs 代码如下：

```
    public static ObjectContext GetCurrentDbContext()
    {
        ObjectContext context = (ObjectContext)CallContext.GetData
("ChannelContent");
        if (context == null)
        {
            context = new ChannelContainer();
            context.Connection.Open();
            CallContext.SetData("ChannelContent", context);
        }
```

```
        return context;
    }
```

通过继承 BaseDll，实现微信用户信息表的数据访问类 WeixinUserInfoDll、推广渠道类型表的数据访问类 ChannelTypeDll、推广渠道表的数据访问类 ChannelDll，以及推广渠道二维码扫描记录表的数据访问类 ChannelScanDll。由于基类已实现了通用的数据访问方法，具体的数据访问类中只需实现该数据表特殊的数据访问方法即可。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Sample_5.ORM;
namespace Sample_5.DAL
{
    /// <summary>
    /// 微信用户信息表的数据访问类
    /// </summary>
    public class WeixinUserInfoDll : BaseDll<WeixinUserInfo>
    {
        /// <summary>
        /// 根据 OpenId 删除微信用户信息
        /// </summary>
        /// <param name="openId">微信用户 OpenId</param>
        /// <returns></returns>
        public bool DeleteByOpenId(string openId)
        {
            var entity = LoadEntity(p => p.OpenId == openId);
            if (entity != null)
            {
                return DeleteEntity(entity);
            }
            else
                return false;
        }
        /// <summary>
        /// 插入微信用户信息
        /// </summary>
        /// <param name="userInfo"></param>
        /// <returns></returns>
```

```
        public void Insert(WeixinUserInfo userInfo)
        {
            AddEntity(userInfo);
        }
        /// <summary>
        /// 更新微信用户信息
        /// </summary>
        /// <param name="userInfo"></param>
        /// <returns></returns>
        public void Update(WeixinUserInfo userInfo)
        {
            var entity = new WeixinUserInfoDll().LoadEntity(p => p.OpenId ==
userInfo.OpenId);
            userInfo.ID = entity.ID;
            UpdateEntity(userInfo);
        }
    }
    /// <summary>
    /// 推广渠道表的数据访问类
    /// </summary>
    public class ChannelDll : BaseDll<Channel>
    {
        /// <summary>
        /// 是否存在该渠道名称
        /// </summary>
        /// <param name="channelName">渠道名称</param>
        /// <param name="id">渠道 ID</param>
        /// <returns></returns>
        public bool IsExitChannelName(string channelName, int id)
        {
            var entities = LoadEntities(p=>p.Name == channelName);
            if (id > 0)
            {
                entities = entities.Where(e => e.ID != id);
            }
            return entities.Any();
        }
    }
    /// <summary>
    /// 推广渠道类型表的数据访问类
    /// </summary>
```

```
    public class ChannelTypeDll : BaseDll<ChannelType>
    {
    }
    /// <summary>
    /// 推广渠道二维码扫描记录表的数据访问类
    /// </summary>
    public class ChannelScanDll: BaseDll<ChannelScan>
    {
    }
}
```

8.3.2　实现视图实体层

按照三层架构的规范，还需要定义供业务逻辑层和用户界面交互层调用的实体，以及数据实体与视图实体的转换方法。

与 8.3.1 节一样，我们直接使用 6.3.5 节中已实现的数据实体与视图实体的转换接口 IViewModel 及 Object-Object Mapping 工具 AutoMapper 实现视图实体层。

首先定义视图实体所需的渠道扫描类型枚举 ScanType 及性别枚举 Sex。

```
namespace Sample_5.ViewEntity
{
    /// <summary>
    /// 渠道扫描类型
    /// </summary>
    public enum ScanType
    {
        /// <summary>
        /// 关注
        /// </summary>
        Subscribe = 1,
        /// <summary>
        /// 已关注后扫描
        /// </summary>
        Scan
    }
    /// <summary>
    /// 性别
    /// </summary>
    public enum Sex
```

```
        {
            /// <summary>
            /// 未知
            /// </summary>
            UnKnown,
            /// <summary>
            /// 男
            /// </summary>
            Male,
            /// <summary>
            /// 女
            /// </summary>
            Female
        }
    }
```

然后分别实现微信用户信息表、推广渠道类型表、推广渠道表及推广渠道二维码扫描记录表的视图实体。

```
    using System;
    using System.ComponentModel.DataAnnotations;
    using Sample_5.ORM;

    namespace Sample_5.ViewEntity
    {
        /// <summary>
        /// 微信用户信息
        /// </summary>
        [Serializable]
        public class WeixinUserInfoEntity : IViewModel<WeixinUserInfoEntity,
WeixinUserInfo>
        {
            static WeixinUserInfoEntity()
            {
                var map = AutoMapper.Mapper.CreateMap<WeixinUserInfo,
WeixinUserInfoEntity>();
                map.ForMember(e => e.Sex, d => d.MapFrom(n => (Sex)n.Sex));
                var m1 = AutoMapper.Mapper.CreateMap<WeixinUserInfoEntity,
WeixinUserInfo>();
                m1.ForMember(e => e.Sex, d => d.MapFrom(n => (short)n.Sex));
            }
            /// <summary>
```

```
        /// ID
        /// </summary>
        [Key]
        public int ID { get; set; }
        /// <summary>
        /// 微信用户的 OpenId
        /// </summary>
        public string OpenId { get; set; }
        /// <summary>
        /// 昵称
        /// </summary>
        public string NickName { get; set; }
        /// <summary>
        /// 普通用户的头像链接
        /// </summary>
        public string HeadImgUrl { get; set; }
        /// <summary>
        /// 普通用户的语言，简体中文为 zh_CN
        /// </summary>
        public string Language { get; set; }
        /// <summary>
        /// 性别
        /// </summary>
        public Sex Sex { get; set; }
        /// <summary>
        /// 用户所在城市
        /// </summary>
        public string City { get; set; }
        /// <summary>
        /// 用户所在省份
        /// </summary>
        public string Province { get; set; }
        /// <summary>
        /// 用户所在国家
        /// </summary>
        public string Country { get; set; }
        /// <summary>
        /// 微信服务器保存的用户关注时间，为时间戳。如果用户曾多次关注，则取最后关注时间
        /// </summary>
        public long Subscribe_time { get; set; }
        /// <summary>
```

```
            /// 项目中实际使用的用户关注时间
            /// </summary>
            public DateTime SubscribeTime
            {
                get
                {
                    //微信服务器保存的用户关注时间时间戳，为从1970年1月1日0时起到用户
关注时所经过的毫秒数
                    //需要进行计算才能正确转换为C#中DateTime类型的时间
                    DateTime dtStart = TimeZone.CurrentTimeZone.ToLocalTime(new
DateTime(1970, 1, 1));
                    long lTime = long.Parse(Subscribe_time + "0000000");
                    TimeSpan toNow = new TimeSpan(lTime);
                    DateTime dtResult = dtStart.Add(toNow);
                    return dtResult;
                }
            }
            public WeixinUserInfoEntity GetViewModel(WeixinUserInfo entity)
            {
              return AutoMapper.Mapper.Map<WeixinUserInfo, WeixinUserInfoEntity>
(entity);
            }
            public WeixinUserInfo GetDataEntity(WeixinUserInfoEntity entity)
            {
              return AutoMapper.Mapper.Map<WeixinUserInfoEntity, WeixinUserInfo>
(entity);
            }
        }
        /// <summary>
        /// 渠道
        /// </summary>
        [Serializable]
        public class ChannelEntity : IViewModel<ChannelEntity, Channel>
        {
            static ChannelEntity()
            {
                AutoMapper.Mapper.CreateMap<Channel, ChannelEntity>();
                AutoMapper.Mapper.CreateMap<ChannelEntity, Channel>();
            }
            /// <summary>
            /// ID
```

```
        /// </summary>
        [Key]
        public int ID { get; set; }
        /// <summary>
        /// 场景值 ID，临时二维码时为 32 位非 0 整型，永久二维码时最大值为 100 000（目
前参数只支持 1～100 000）
        /// </summary>
        public int SceneId { get; set; }
        /// <summary>
        /// 渠道名称
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// 所属渠道类型 ID
        /// </summary>
        public int ChannelTypeId { get; set; }
        /// <summary>
        /// 渠道二维码
        /// </summary>
        public string Qrcode { get; set; }
        public ChannelEntity GetViewModel(Channel entity)
        {
            return AutoMapper.Mapper.Map<Channel, ChannelEntity>(entity);
        }
        public Channel GetDataEntity(ChannelEntity entity)
        {
            return AutoMapper.Mapper.Map<ChannelEntity, Channel>(entity);
        }
    }
    /// <summary>
    /// 渠道类型
    /// </summary>
    [Serializable]
    public class ChannelTypeEntity : IViewModel<ChannelTypeEntity,
ChannelType>
    {
        static ChannelTypeEntity()
        {
            AutoMapper.Mapper.CreateMap<ChannelType, ChannelTypeEntity>();
            AutoMapper.Mapper.CreateMap<ChannelTypeEntity, ChannelType>();
        }
```

```
            /// <summary>
            /// ID
            /// </summary>
            [Key]
            public int ID { get; set; }
            /// <summary>
            /// 渠道类型名称
            /// </summary>
            public string Name { get; set; }
            public ChannelTypeEntity GetViewModel(ChannelType entity)
            {
                return AutoMapper.Mapper.Map<ChannelType, ChannelTypeEntity>
(entity);
            }
            public ChannelType GetDataEntity(ChannelTypeEntity entity)
            {
                return AutoMapper.Mapper.Map<ChannelTypeEntity, ChannelType>
(entity);
            }
        }
        /// <summary>
        /// 渠道二维码扫描记录
        /// </summary>
        [Serializable]
        public class ChannelScanEntity : IViewModel<ChannelScanEntity,
ChannelScan>
        {
            static ChannelScanEntity()
            {
                var map = AutoMapper.Mapper.CreateMap<ChannelScan,
ChannelScanEntity>();
                map.ForMember(e => e.ScanType, d => d.MapFrom(n => (ScanType)
n.ScanType));
                var m1 = AutoMapper.Mapper.CreateMap<ChannelScanEntity,
ChannelScan>();
                m1.ForMember(e => e.ScanType, d => d.MapFrom(n => (short)
n.ScanType));
            }
            /// <summary>
            /// ID
            /// </summary>
```

```
        [Key]
        public int ID { get; set; }
        /// <summary>
        /// 扫描微信用户的 OpenId
        /// </summary>
        public string OpenId { get; set; }
        /// <summary>
        /// 扫描类型
        /// </summary>
        public ScanType ScanType { get; set; }
        /// <summary>
        /// 扫描时间
        /// </summary>
        public DateTime ScanTime { get; set; }
        /// <summary>
        /// 所属渠道
        /// </summary>
        public int ChannelId { get; set; }
        public ChannelScanEntity GetViewModel(ChannelScan entity)
        {
            return AutoMapper.Mapper.Map<ChannelScan, ChannelScanEntity>
(entity);
        }
        public ChannelScan GetDataEntity(ChannelScanEntity entity)
        {
            return AutoMapper.Mapper.Map<ChannelScanEntity, ChannelScan>
(entity);
        }
    }
}
```

由于在前端页面中显示的推广渠道二维码扫描记录需要包含微信用户个人信息。为了方便前端调用，还需要定义一个用于页面显示的推广渠道二维码扫描记录视图实体 ChannelScanDisplayEntity。

```
using System;
namespace Sample_5.ViewEntity
{
    /// <summary>
    /// 用于前端口显示的渠道扫描记录
    /// </summary>
    [Serializable]
```

```
    public class ChannelScanDisplayEntity
    {
        /// <summary>
        /// 渠道扫描记录
        /// </summary>
        public ChannelScanEntity ScanEntity { get; set; }
        /// <summary>
        /// 扫描渠道的微信用户信息
        /// </summary>
        public WeixinUserInfoEntity UserInfoEntity { get; set; }
    }
}
```

8.3.3 同步微信个人用户信息

当有微信用户扫描二维码时，微信服务器传递的事件中仅包含微信用户的OpenId。要获取微信用户的信息，需要调用微信公众平台的“用户信息接口”。

在理想情况下，仅需要在扫描事件处理程序中判断 OpenId 是否存在于WeixinUserInfo 表中，如果不存在，则从微信服务器获取该 OpenId 对应的微信用户信息保存到 WeixinUserInfo 表中。

但微信用户可能在任何时间修改自己的基本资料，以及取消对微信公众号的关注。如果仅在用户扫描的第一次将微信用户信息保存到本地，则很可能导致微信用户的最新信息与本地数据库中的信息不一致，造成系统中显示的微信用户信息是过时和错误的。

为了解决上述问题，我们需要定时同步策略来实现微信用户信息的获取。具体的策略思路描述如下：

1．在 ASP.NET 应用程序启动后，当网页第一次被访问时，开启一个进程内全局唯一的微信用户信息同步线程 SynchronizeWeixinUserThread。

2．由于网页可能被同时访问，需要使用单例模式确保同步线程 SynchronizeWeixinUserThread 不会启动多次。

3．SynchronizeWeixinUserThread 线程启动后，每隔一定时间，执行一次同步方法 SynchronizeWeixinUser。

4. SynchronizeWeixinUser 同步方法首先调用微信公众平台接口，获取微信公众号所有关注者的 OpenId。然后逐一判断数据库中是否存在该 OpenId，如果数据库中不存在，则说明是在上一次同步结束后新关注的用户，需要获取用户信息后插入数据库；如果数据库中存在，则需要将最新的用户信息更新到数据库。同时，还需要找出在数据库中而不在通过接口获取到的关注者列表中的 OpenId，这些 OpenId 对应的用户已取消关注，需要从数据库中删除。

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Threading;
using System.Web;
using Sample_5.DAL;
using Sample_5.ORM;
using Sample_5.ViewEntity;
using Senparc.Weixin.MP;
using Senparc.Weixin.MP.AdvancedAPIs;
using Senparc.Weixin.MP.CommonAPIs;
namespace Sample_5.BLL
{
    /// <summary>
    /// 同步微信用户信息
    /// 静态类，保证跨线程全局唯一的
    /// </summary>
    public static class WeixinUserInfoSynchronize
    {
        private delegate void GetExecute(WeixinUserInfo userInfo);
        /// <summary>
        /// 同步微信用户信息线程
        /// </summary>
        private static Thread SynchronizeWeixinUserThread = null;
        /// <summary>
        /// 锁
        /// </summary>
        private static object lockSingal = new object();

        /// <summary>
        /// 开启同步微信用户信息线程
        /// 单例模式
```

```
        /// </summary>
        public static void Synchronize()
        {
            if (SynchronizeWeixinUserThread == null)
            {
                lock(lockSingal)
                {
                    if (SynchronizeWeixinUserThread == null)
                    {
                        // 开启同步微信用户信息的后台线程
                        ThreadStart start = new ThreadStart
(SynchronizeWeixinUserCircle);
                        SynchronizeWeixinUserThread = new Thread(start);
                        SynchronizeWeixinUserThread.Start();
                    }
                }
            }
        }
        /// <summary>
        /// 每隔 60 秒执行一次微信用户信息同步方法
        /// </summary>
        private static void SynchronizeWeixinUserCircle()
        {
            try
            {
                SynchronizeWeixinUser();
                Thread.Sleep(60000);
            }
            catch { }
        }
        /// <summary>
        /// 微信用户信息同步方法
        /// </summary>
        /// <returns></returns>
        private static void SynchronizeWeixinUser()
        {
            OpenIdResultJson weixinOpenIds = GetAllOpenIds();
            //获取已同步到数据库中的微信用户的 OpenId
            List<string> dataOpenList = new WeixinUserInfoDll().LoadEntities
(p => p.ID > 0).Select(e => e.OpenId).ToList();
            List<string> insertOpenIdList = new List<string>();
```

```
            List<string> updateOpenIdList = new List<string>();
            List<string> deleteOpenIdList = new List<string>();
            //判断每个微信用户需要执行的操作
            for (int index = 0; index < weixinOpenIds.data.openid.Count;
index++)
            {
                var weixinOpenId = weixinOpenIds.data.openid[index];
                var user = dataOpenList.Find(e => e == weixinOpenId);
                if (user == null)
                {
                    //不存在数据库中的，插入
                    insertOpenIdList.Add(weixinOpenId);
                }
                else
                {
                    //已存在数据库中的，修改
                    updateOpenIdList.Add(weixinOpenId);
                }
            }
            //已取消关注该微信公众号的，删除
            insertOpenIdList.ForEach(e => dataOpenList.Remove(e));
            updateOpenIdList.ForEach(e => dataOpenList.Remove(e));
            deleteOpenIdList.AddRange(dataOpenList);
            //插入失败的 openId 列表，用于失败重试
            List<string> failedInsert = new List<string>();
            //修改失败的 openId 列表，用于失败重试
            List<string> failedUpdate = new List<string>();
            //插入新的微信用户
            foreach (var openId in insertOpenIdList)
            {
                ExecuteWeixinUser(openId, new WeixinUserInfoDll().Insert,
failedInsert);
            }
            //更新已有微信用户
            foreach (var openId in updateOpenIdList)
            {
                ExecuteWeixinUser(openId, new WeixinUserInfoDll().Update,
failedUpdate);
            }
            if (deleteOpenIdList.Count > 0)
            {
```

```
                //删除已取消关注该微信公众号的微信用户
                foreach (var openId in deleteOpenIdList)
                {
                    new WeixinUserInfoDll().DeleteByOpenId(openId);
                }
            }
            //插入失败，重试一次
            if (failedInsert.Count > 0)
            {
                List<string> fail = new List<string>();
                foreach (var openId in failedInsert)
                {
                    ExecuteWeixinUser(openId, new WeixinUserInfoDll().Insert,
fail);
                }
            }
            //更新失败，重试一次
            if (failedUpdate.Count > 0)
            {
                List<string> fail = new List<string>();
                foreach (var openId in failedInsert)
                {
                    ExecuteWeixinUser(openId, new WeixinUserInfoDll().Update,
fail);
                }
            }
        }
        /// <summary>
        /// 获取所有关注者的OpenId信息
        /// </summary>
        /// <returns></returns>
        private static OpenIdResultJson GetAllOpenIds()
        {
            string accessToken = AccessTokenContainer.TryGetToken
(ConfigurationManager.AppSettings["appID"], ConfigurationManager.
AppSettings["appsecret"]);
            OpenIdResultJson openIdResult = User.List(accessToken, null);
            while (!string.IsNullOrWhiteSpace(openIdResult.next_openid))
            {
             OpenIdResultJson tempResult = User.List(accessToken, openIdResult.
next_openid);
```

```
                openIdResult.next_openid = tempResult.next_openid;
                if (tempResult.data != null && tempResult.data.openid != null)
                {
                    openIdResult.data.openid.AddRange(tempResult.data.
openid);
                }
            }
            return openIdResult;
        }

        /// <summary>
        /// 获取 openId 对应的用户信息并存入数据库
        /// </summary>
        /// <param name="openId">微信用户 OpenId</param>
        /// <param name="execute">修改、删除或插入操作</param>
        /// <param name="failList">未成功获取到用户信息的 OpenId 列表</param>
        private static void ExecuteWeixinUser(string openId, GetExecute
execute, List<string> failList)
        {
            string accessToken = AccessTokenContainer.TryGetToken
(ConfigurationManager.AppSettings["appID"],
ConfigurationManager.AppSettings["appsecret"]);
            var userInfo = User.Info(accessToken, openId);
            if (userInfo.errcode != ReturnCode.请求成功)
            {
                failList.Add(openId);
            }
            else
            {
                WeixinUserInfo entity = new WeixinUserInfo()
                {
                    City = userInfo.city,
                    Province = userInfo.province,
                    Country = userInfo.country,
                    HeadImgUrl = userInfo.headimgurl,
                    Language = userInfo.language,
                    Subscribe_time = userInfo.subscribe_time,
                    Sex = (Int16)userInfo.sex,
                    NickName = userInfo.nickname,
                    OpenId = userInfo.openid
```

```
                };
                execute(entity);
            }
        }
    }
}
```

8.3.4 实现业务逻辑层

在实现业务逻辑的过程中 ，有几点需要注意的地方。首先，WeixinUserInfoBll需要使用静态构造函数，以确保构造函数只被执行一次，避免微信用户同步线程被开启多次。

在微信用户扫描二维码时，微信服务器会将扫描事件发送到微信公众号服务器上，而微信服务器要求微信公众号服务器必须在 5 秒内返回响应结果。微信服务器在 5 秒内收不到响应会断掉连接，并且重新发起请求，总共重试三次。为了避免微信公众号服务器重复接收到同一条扫描事件，造成数据重复，导致统计失真，需要将保存扫描记录的操作放到线程池中异步执行，确保微信公众号服务器尽快返回响应结果给微信服务器。

在新增推广渠道时，我们需要确保新增推广渠道的场景值 ID 不与任何已存在的推广渠道重复，这样分配每个推广渠道的二维码才具有唯一性。否则，将出现用户只进行了一次扫描，但被同时算做多个推广渠道的推广效果的情况。同时，为了避免每次下载推广渠道的二维码，都去请求微信服务器，在新增推广渠道时，需要将场景值 ID 对应的二维码保存到本地，并在数据库中保存二维码保存路径。

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Linq;
using System.Threading;
using System.Web;
using Sample_5.DAL;
using Sample_5.ORM;
using Sample_5.ViewEntity;
namespace Sample_5.BLL
{
    public class WeixinUserInfoBll
```

```
        {
            /// <summary>
            /// 静态构造函数，确保微信用户同步线程只被开启一次
            /// </summary>
            static WeixinUserInfoBll()
            {
                WeixinUserInfoSynchronize.Synchronize();
            }
            /// <summary>
            /// 获取微信用户信息列表
            /// </summary>
            /// <returns></returns>
            public List<WeixinUserInfoEntity> GetEntities()
            {
              var entities = new WeixinUserInfoDll().LoadEntities(p => p.OpenId !=
"").ToList();
                var viewEntity = new WeixinUserInfoEntity();
                return entities.Select(p => viewEntity.GetViewModel(p)).ToList();
            }
        }
        public class ChannelTypeBll
        {
            /// <summary>
            /// 获取渠道类型列表
            /// </summary>
            /// <returns></returns>
            public List<ChannelTypeEntity> GetEntities()
            {
                var entities = new ChannelTypeDll().LoadEntities(p => p.ID >
0).ToList();
                var viewEntity = new ChannelTypeEntity();
              return entities.Select(p => viewEntity.GetViewModel(p)).ToList();
            }
            /// <summary>
            /// 根据 ID 获取渠道类型
            /// </summary>
            /// <param name="id">渠道类型 ID</param>
            /// <returns></returns>
            public ChannelTypeEntity GetEntityById(int id)
            {
                var entity = new ChannelTypeDll().LoadEntity(p => p.ID == id);
```

```
            var viewEntity = new ChannelTypeEntity();
            return viewEntity.GetViewModel(entity);
        }
        /// <summary>
        /// 添加或修改渠道类型
        /// </summary>
        /// <param name="viewEntity">渠道类型实体</param>
        /// <returns></returns>
        public bool UpdateOrInsertEntity(ChannelTypeEntity viewEntity)
        {
            var entity = viewEntity.GetDataEntity(viewEntity);
            if (entity.ID > 0)
            {
                return new ChannelTypeDll().UpdateEntity(entity);
            }
            else
            {
                return new ChannelTypeDll().AddEntity(entity).ID > 0;
            }
        }
        /// <summary>
        /// 根据 ID 删除渠道类型
        /// </summary>
        /// <param name="id">渠道类型 ID</param>
        /// <returns></returns>
        public bool DeleteEntityById(int id)
        {
            var entity = new ChannelTypeDll().LoadEntity(p => p.ID == id);
            return new ChannelTypeDll().DeleteEntity(entity);
        }
    }
    public class ChannelScanBll
    {
        /// <summary>
        /// 保存扫描记录
        /// </summary>
        /// <param name="openId">微信用户 OpenId</param>
        /// <param name="sceneId">扫描的二维码的参数</param>
        /// <param name="scanType">扫描类型</param>
        public void SaveScan(string openId, int sceneId, ScanType scanType)
        {
```

```
                //微信公众平台要求微信公众号服务器必须在 5 秒内返回相应结果，否则会重新发
送请求，一共重试三次
                //为了避免微信公众号服务器重复接收到同一条扫描记录，造成数据重复，导致统
计失真，这里将保存扫描记录的操作放到线程池中异步执行，尽快返回相应结果给微信服务器
            ThreadPool.QueueUserWorkItem(e =>
            {
              int channelId = new ChannelBll().GetChannelIdBySceneId(sceneId);
               if (channelId <= 0)
               {
                  return;
               }
               ChannelScanEntity entity = new ChannelScanEntity()
               {
                  ChannelId = channelId,
                  ScanTime = DateTime.Now,
                  OpenId = openId,
                  ScanType = scanType
               };
               new ChannelScanDll().AddEntity(entity.GetDataEntity(entity));
            });
         }
         /// <summary>
         /// 获取渠道的扫描记录
         /// </summary>
         /// <param name="channelId">渠道 ID</param>
         /// <returns></returns>
        public List<ChannelScanDisplayEntity> GetChannelScanList(int channelId)
         {
            //获取渠道扫描记录
           var entities = new ChannelScanDll().LoadEntities(p => p.ChannelId
== channelId).ToList();
            var viewEntity = new ChannelScanEntity();
            var result = entities.Select(p => new ChannelScanDisplayEntity()
{ ScanEntity = viewEntity.GetViewModel(p) }).ToList();
            //获取每条渠道扫描记录对应的微信用户信息
            var openIds = result.Select(p=>p.ScanEntity.OpenId).ToArray();
            //在渠道扫描记录中包含微信用户信息，便于前端页面显示
            var userinfoEntities = new WeixinUserInfoDll().LoadEntities(p =>
openIds.Contains(p.OpenId)).ToList();
            var userinfoViewEntity = new WeixinUserInfoEntity();
```

```
            var userinfoViewEnities = userinfoEntities.Select(p =>
userinfoViewEntity.GetViewModel(p)).ToList();
            result.ForEach(e=>{
               e.UserInfoEntity = userinfoViewEnities.Where(p => p.OpenId ==
e.ScanEntity.OpenId).FirstOrDefault();
            });
            return result;
        }
    }
    public class ChannelBll
    {
        /// <summary>
        /// 获取渠道列表
        /// </summary>
        /// <returns></returns>
        public List<ChannelEntity> GetEntities()
        {
          var entities = new ChannelDll().LoadEntities(p => p.ID > 0).ToList();
           var viewEntity = new ChannelEntity();
          return entities.Select(p => viewEntity.GetViewModel(p)).ToList();
        }
        /// <summary>
        /// 根据 ID 获取渠道
        /// </summary>
        /// <param name="id">渠道 ID</param>
        /// <returns></returns>
        public ChannelEntity GetEntityById(int id)
        {
           var entity = new ChannelDll().LoadEntity(p => p.ID == id);
           var viewEntity = new ChannelEntity();
           return viewEntity.GetViewModel(entity);
        }
        /// <summary>
        /// 添加或修改渠道
        /// </summary>
        /// <param name="viewEntity">渠道实体</param>
        /// <returns></returns>
        public bool UpdateOrInsertEntity(ChannelEntity viewEntity)
        {
           if (viewEntity.ID > 0)
           {
```

```
                var entity=viewEntity.GetDataEntity(viewEntity);
                var dbEntity = new ChannelDll().LoadEntity(p => p.ID ==
entity.ID);
                entity.SceneId = dbEntity.SceneId;
                entity.Qrcode = dbEntity.Qrcode;
                return new ChannelDll().UpdateEntity(entity);
            }
            else
            {
                //新增渠道时，需要获取渠道的二维码
                GetQrcode(viewEntity);
                var entity = viewEntity.GetDataEntity(viewEntity);
                return new ChannelDll().AddEntity(entity).ID > 0;
            }
        }
        /// <summary>
        /// 根据 ID 删除渠道
        /// </summary>
        /// <param name="id">渠道 ID</param>
        /// <returns></returns>
        public bool DeleteEntityById(int id)
        {
            var entity = new ChannelDll().LoadEntity(p => p.ID == id);
            return new ChannelDll().DeleteEntity(entity);
        }
        /// <summary>
        /// 根据 SceneId 获取二维码 ID
        /// </summary>
        /// <param name="sceneId">扫描的二维码的参数</param>
        /// <returns></returns>
        public int GetChannelIdBySceneId(int sceneId)
        {
            var entity = new ChannelDll().LoadEntity(p=>p.SceneId == sceneId);
            return entity == null ? 0 : entity.ID;
        }
        /// <summary>
        /// 判断渠道名称是否存在
        /// </summary>
        /// <param name="channelName">渠道名称</param>
        /// <param name="id">渠道 ID</param>
        /// <returns></returns>
```

```
        public bool IsExitChannelName(string channelName, int id)
        {
            return new ChannelDll().IsExitChannelName(channelName, id);
        }
        /// <summary>
        /// 获取渠道的二维码
        /// </summary>
        /// <param name="channelName">渠道实体</param>
        /// <returns></returns>
        private void GetQrcode(ChannelEntity entity)
        {
            //获取微信公众平台接口访问凭据
            string accessToken = AccessTokenContainer.TryGetToken
(ConfigurationManager.AppSettings["appID"], ConfigurationManager.AppSettings
["appsecret"]);
            //找出一个未被使用的场景值 ID，确保不同渠道使用不同的场景值 ID
         int scenid = GetNotUsedSmallSceneId();
            if (scenid <= 0 || scenid > 100000)
            {
                throw new Exception("抱歉，您的二维码已经用完，请删除部分后重新添加");
            }
          CreateQrCodeResult createQrCodeResult = QrCode.Create(accessToken,
0, scenid);
            if (!string.IsNullOrEmpty(createQrCodeResult.ticket))
            {
                using (MemoryStream stream = new MemoryStream())
                {
                    //根据 ticket 获取二维码
                    QrCode.ShowQrCode(createQrCodeResult.ticket, stream);
                    //将获取到的二维码图片转换为 Base64String 格式
                    byte[] imageBytes = stream.ToArray();
                 string base64Image = System.Convert.ToBase64String(imageBytes);
                    //由于 SqlServerCompact 数据库限制最长字符 4000，本测试项目将二
维码保存到磁盘，在正式项目中可直接保存到数据库
                    string imageFile = "QrcodeImage" + scenid.ToString() +
".img";
                    File.WriteAllText(System.Web.HttpContext.Current.Server.
MapPath("~/App_Data/") + imageFile, base64Image);
                    entity.Qrcode = imageFile;
                    entity.SceneId = scenid;
                }
```

```
            }
            else
            {
                throw new Exception("抱歉! 获取二维码失败");
            }
        }
        /// <summary>
        /// 找出没有用的最小 SceneId
        /// </summary>
        /// <returns></returns>
        private int GetNotUsedSmallSceneId()
        {
            var listSceneId = new ChannelDll().LoadEntities(p => p.ID >
0).Select(p => p.SceneId).OrderBy(p => p);
            for (int i = 1; i <= 100000; i++)
            {
                var sceneId = listSceneId.Any(e => e == i);
                if (!sceneId)
                {
                    return i;
                }
            }
            return 0;
        }
    }
}
```

8.3.5　推广渠道管理系统后台实现

推广渠道管理系统后台功能主要包括：

1. 接收二维码扫描事件

2. 推广渠道类型管理

3. 推广渠道管理

4. 推广渠道二维码扫描记录列表

与 6.7 节一样，页面前端开发框架同样是使用 Bootstrap，接下来分别实现上述功能。

1. 接收二维码扫描事件

对未关注用户和已关注用户扫描二维码，微信服务器会发送不同的扫描事件。未关注用户扫描二维码并关注微信公众号后，微信服务器发送的事件类型为“关注事件”，而已关注用户扫描二维码后，微信服务器发送的事件类型为“扫描带参数二维码事件”。

因此，需要在 CustomMessageHandler.cs 中，分别重写处理扫描请求的方法 OnEvent_ScanRequest 及处理关注请求的方法 OnEvent_SubscribeRequest 来保存扫描记录。

```
    /// <summary>
    /// 处理扫描请求
    /// 用户扫描带场景值二维码时，如果用户已经关注公众号，则微信会将带场景值扫描事件推送给开发者
    /// </summary>
    /// <returns></returns>
    protected override IResponseMessageBase OnEvent_ScanRequest(RequestMessageEvent_Scan requestMessage)
    {
        int sceneId = 0;
        int.TryParse(requestMessage.EventKey, out sceneId);
        if (sceneId > 0)
        {
            new ChannelScanBll().SaveScan(requestMessage.FromUserName, sceneId, ScanType.Scan);
        }
        var responseMessage = CreateResponseMessage<ResponseMessageText>();
        responseMessage.Content = "扫描已记录";
        return responseMessage;
    }
    /// <summary>
    /// 处理关注请求
    /// 用户扫描带场景值二维码时，如果用户还未关注公众号，则用户可以关注公众号，关注后微信会将带场景值关注事件推送给开发者
    /// </summary>
    /// <returns></returns>
    protected override IResponseMessageBase OnEvent_SubscribeRequest(RequestMessageEvent_Subscribe requestMessage)
    {
        if (!string.IsNullOrWhiteSpace(requestMessage.EventKey))
```

```
        {
            string sceneIdstr = requestMessage.EventKey.Substring(8);
            int sceneId = 0;
            int.TryParse(sceneIdstr, out sceneId);
            if (sceneId > 0)
            {
                new ChannelScanBll().SaveScan(requestMessage.FromUserName, sceneId,
ScanType.Subscribe);
            }
        }
        var responseMessage = CreateResponseMessage<ResponseMessageText>();
        responseMessage.Content = "扫描已记录";
        return responseMessage;
    }
```

2. 推广渠道类型管理

推广渠道类型管理主要是对推广渠道类型的修改、新增、删除及列表显示等。

新增与修改页面可以合并为一个页面，后端根据前端传递的表单数据中推广渠道类型是否大于零，来判断是新增推广渠道类型还是修改已有推广渠道类型。新建一个 Web 窗体 ChannelTypeEdit，实现推广渠道类型新增与修改。ChannelTypeEdit 的后端代码如下：

```
    public partial class ChannelTypeEdit : System.Web.UI.Page
    {
        /// <summary>
        /// 新增或修改渠道类型
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack)
            {
                //修改渠道类型，首先获取渠道类型数据
                int id;
                if (int.TryParse(Request.QueryString["id"], out id))
                {
                    ViewState["channelType"] = new ChannelTypeBll().GetEntityById
(id);
```

```
            }
        }
        else
        {
            //将渠道类型新增或修改的数据保存到数据库
            var entity = new ChannelTypeEntity()
            {
              ID = Request.Form["ID"] == null ? 0 : int.Parse(Request.Form["ID"]),
                Name = Request.Form["Name"]
            };
            new ChannelTypeBll().UpdateOrInsertEntity(entity);
            //回到渠道类型列表页面
            Response.Redirect("ChannelTypeList.aspx");
            Response.End();
        }
    }
}
```

ChannelTypeEdit 页面代码如下：

```
    <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ChannelTypeEdit.
aspx.cs"Inherits="Sample_5.ChannelTypeEdit" %>
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head id="Head1" runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
       <title>修改渠道类型</title>
    </head>
    <body>
       <form id="Form1" class="form-horizontal" type="post" runat="server">
          <%if (ViewState["channelType"] != null)
          {%>
             <%-- 修改表单 --%>
            <input type="hidden" name="ID" value ="<%=(ViewState["channelType"]
as Sample_5.ViewEntity.ChannelTypeEntity).ID%>" />
             <div class="control-group">
                <label class="control-label" for="Name"><strong>渠道类型名称
</strong></label>
                <div class="controls">
                  <input type="text" name="Name" value="<%=(ViewState
["channelType"] as Sample_5.ViewEntity.ChannelTypeEntity). Name%>"/>
                </div>
```

```
            </div>
         <%}else{%>
            <%-- 新增表单 --%>
            <div class="control-group">
               <label class="control-label" for="Name"><strong>渠道类型名称
</strong></label>
               <div class="controls">
                  <input type="text" name="Name"/>
               </div>
            </div>
         <%}%>
         <%-- 提交按钮 --%>
         <button  class="btn  btn-large  btn-primary"  type="submit"> 保 存
</button>
      </form>
   </body>
   </html>
```

推广渠道类型列表的 Web 窗体为 ChannelTypeList，可以对列表中的每个推广渠道类型进行编辑和删除操作。

```
   public partial class ChannelTypeList : System.Web.UI.Page
   {
      /// <summary>
      /// 渠道类型列表
      /// </summary>
      /// <param name="sender"></param>
      /// <param name="e"></param>
      protected void Page_Load(object sender, EventArgs e)
      {
         //获取渠道类型列表数据
         ViewState["ChannelTypeList"] = new ChannelTypeBll().GetEntities();
      }
   }
```

ChannelTypeList 页面代码如下：

```
   <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ChannelTypeList.
aspx.cs" Inherits="Sample_5.ChannelTypeList" %>
   <!DOCTYPE html>
   <html xmlns="http://www.w3.org/1999/xhtml">
   <head id="Head1" runat="server">
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
        <title>渠道类型列表</title>
    </head>
    <!-- #Bootstrap -->
    <link href="Css/bootstrap.min.css" rel="stylesheet" />
    <script src="Scripts/bootstrap.min.js"></script>
    <!-- #Jquery -->
    <script src="Scripts/jquery-1.9.1.min.js"></script>
    <body>
       <div class="container">
          <form id="Form1" class="form-horizontal" type="post" runat="server">
             <h2 class="form-signin-heading">渠道类型列表</h2> <a class=
"btnGrayS vm doaddit" href="ChannelTypeEdit.aspx">添加</a> <a class="btnGrayS
vm doaddit" href="ChannelList.aspx">渠道列表</a>
             <table id="listTable" class="table table-bordered table-hover
dataTable" style="width: auto">
                <thead>
                   <tr>
                      <th style="border-color: #ddd; color: Black">ID
                      </th>
                      <th style="border-color: #ddd; color: Black">渠道类型
名称
                      </th>
                      <th style="border-color: #ddd; color: Black">操作
                      </th>
                   </tr>
                </thead>
                <tbody>
                 <% foreach (Sample_5.ViewEntity.ChannelTypeEntity channelType
in (ViewState["ChannelTypeList"] as List<Sample_5.ViewEntity.ChannelTypeEntity>))
                    { %>
                       <tr>
                          <td><%= channelType.ID%>
                          </td>
                          <td><%= channelType.Name%>
                          </td>
                          <td>
                             <p>
                                <a class="btnGrayS vm doaddit"href="
ChannelTypeEdit.aspx?id=<%= channelType.ID%>">编辑</a>
                                <a href="javascript:void(0)" data="<%=
channelType.ID%>" class="dodelit deleteChannelType">删除</a>
```

```
                        </p>
                    </td>
                </tr>
            <% } %>
        </tbody>
    </table>
</form>
</div>
<script>
    $(function () {
        $(".deleteChannelType").click(function () {
            if (confirm("确定删除吗? ")) {
                var id = $(this).attr("data");
                $.ajax({
                    url: "ChannelTypeDelete.aspx?id=" + id,
                    type: "get",
                    success: function (repText) {
                        if (repText && repText == "True") {
                            window.location.reload();
                        } else {
                            alert(repText.ErrorMsg);
                        }
                    },
                    complete: function (xhr, ts) {
                        xhr = null;
                    }
                });
            }
        });
    });
</script>
</body>
</html>
```

删除操作对应的 ChannelTypeDelete.aspx 的代码如下：

```
public partial class ChannelTypeDelete : System.Web.UI.Page
{
    /// <summary>
    /// 删除渠道类型
    /// </summary>
    /// <param name="sender"></param>
```

```
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        //获取渠道类型 ID
        int id;
        if (int.TryParse(Request.QueryString["id"], out id))
        {
            //删除渠道类型并返回删除结果
            bool result = new ChannelTypeBll().DeleteEntityById(id);
            Response.Write(result.ToString());
            Response.End();
        }
    }
}
```

3. 推广渠道管理

推广渠道管理主要包含修改现有推广渠道、添加新的推广渠道、删除推广渠道、列表显示已有推广渠道及下载推广渠道的二维码等。

ChannelEdit 为新增与修改推广渠道的 Web 窗体。

```
public partial class ChannelEdit : System.Web.UI.Page
{
    /// <summary>
    /// 新增或修改渠道
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            //获取渠道类型列表数据
            ViewState["ChannelTypeList"] = new ChannelTypeBll().GetEntities();
            //修改渠道，首先获取渠道数据
            int id;
            if (int.TryParse(Request.QueryString["id"], out id))
            {
                ViewState["Channel"] = new ChannelBll().GetEntityById(id);
            }
        }
```

```
        else
        {
            //将渠道新增或修改的数据保存到数据库
            var entity = new ChannelEntity()
            {
                ID = Request.Form["ID"] == null ? 0 : int.Parse(Request.Form
["ID"]),
                Name = Request.Form["Name"],
                ChannelTypeId = int.Parse(Request.Form["ChannelTypeId"])
            };
            new ChannelBll().UpdateOrInsertEntity(entity);
            //回到渠道列表页面
            Response.Redirect("ChannelList.aspx");
            Response.End();
        }
    }
}
```

ChannelEdit 的页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ChannelEdit.
aspx.cs" Inherits="Sample_5.ChannelEdit" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>修改渠道</title>
</head>
<body>
    <form id="Form1" class="form-horizontal" type="post" runat="server">
        <%if (ViewState["channel"] != null)
        {%>
            <%-- 修改表单 --%>
            <input type="hidden" name="ID" value ="<%=(ViewState["channel"]
as Sample_5.ViewEntity.ChannelEntity).ID%>" />
            <div class="control-group">
                <label class="control-label" for="Name"><strong>渠道名称
</strong></label>
                <div class="controls">
                 <input type="text" name="Name" value="<%=(ViewState["channel"]
as Sample_5.ViewEntity.ChannelEntity).Name%>"/>
                </div>
```

```
            </div>
            <div class="control-group">
               <label class="control-label" for="ChannelTypeId"><strong>所
属渠道类型</strong></label>
               <div class="controls">
                  <%-- 构造渠道类型下拉框 --%>
                  <select name="ChannelTypeId">
                <% foreach (Sample_5.ViewEntity.ChannelTypeEntity channelType
in (ViewState["ChannelTypeList"] as List<Sample_5.ViewEntity.ChannelTypeEntity>))
                  { %>
                     <%if (channelType.ID == (ViewState["channel"] as
Sample_5.ViewEntity.ChannelEntity).ChannelTypeId)
                      {%>
                     <%-- 设置渠道类型下拉框初始选中项目 --%>
                        <option value="<%=channelType.ID %>" selected=
"selected"><%=channelType.Name %></option>
                     <%}else{%>
                        <option      value="<%=channelType.ID      %>"><%=
channelType.Name %></option>
                     <%}%>
                  <% } %>
                  </select>
               </div>
            </div>
         <%}else{%>
            <%-- 新增表单 --%>
            <div class="control-group">
               <label  class="control-label"  for="Name"><strong> 渠 道 名 称
</strong></label>
               <div class="controls">
                  <input type="text" name="Name"/>
               </div>
            </div>
            <div class="control-group">
               <label class="control-label" for="ChannelTypeId"><strong>所
属渠道类型</strong></label>
               <div class="controls">
                  <%-- 构造渠道类型下拉框 --%>
                  <select name="ChannelTypeId">
                <% foreach (Sample_5.ViewEntity.ChannelTypeEntity channelType
in (ViewState["ChannelTypeList"] as List<Sample_5.ViewEntity.ChannelTypeEntity>))
```

```
                    { %>
                        <option value="<%=channelType.ID %>"><%=
channelType.Name %></option>
                    <% } %>
                    </select>
                </div>
            </div>
        <%}%>
        <%-- 提交按钮 --%>
        <button class="btn btn-large btn-primary" type="submit"> 保 存
</button>
    </form>
</body>
</html>
```

ChannelList 为推广渠道列表显示的 Web 窗体，其中包含修改推广渠道、删除推广渠道、下载二维码等操作。

```
public partial class ChannelList : System.Web.UI.Page
{
    /// <summary>
    /// 渠道列表
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        //获取渠道列表数据
        ViewState["ChannelList"] = new ChannelBll().GetEntities();
    }
}
```

ChannelList 的页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ChannelList.
aspx.cs" Inherits="Sample_5.ChannelList" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>渠道列表</title>
</head>
<!-- #Bootstrap -->
```

```
    <link href="Css/bootstrap.min.css" rel="stylesheet" />
    <script src="Scripts/bootstrap.min.js"></script>
    <!-- #Jquery -->
    <script src="Scripts/jquery-1.9.1.min.js"></script>
    <body>
       <div class="container">
          <form id="Form1" class="form-horizontal" type="post" runat="server">
             <h2 class="form-signin-heading">渠道列表</h2> <a class="btnGrayS
vm doaddit" href="ChannelEdit.aspx">添加</a>
             <table id="listTable" class="table table-bordered table-hover
dataTable" style="width: auto">
                <thead>
                   <tr>
                      <th style="border-color: #ddd; color: Black">ID
                      </th>
                      <th style="border-color: #ddd; color: Black">场景值 ID
                      </th>
                      <th style="border-color: #ddd; color: Black">渠道名称
                      </th>
                      <th style="border-color: #ddd; color: Black">所属渠道
类型 ID
                      </th>
                      <th style="border-color: #ddd; color: Black">操作
                      </th>
                   </tr>
                </thead>
                <tbody>
                   <% foreach (Sample_5.ViewEntity.ChannelEntity channel in
(ViewState["ChannelList"] as List<Sample_5.ViewEntity.ChannelEntity>))
                      { %>
                         <tr>
                            <td><%= channel.ID%>
                            </td>
                            <td><%= channel.SceneId%>
                            </td>
                            <td><%= channel.Name%>
                            </td>
                            <td><%= channel.ChannelTypeId%>
                            </td>
                            <td>
                               <p>
```

```
                                    <a class="btnGrayS vm doaddit"href="
ChannelScanList.aspx?id=<%= channel.ID%>">查看扫描记录</a>
                                    <a class="btnGrayS vm doaddit" href="
QRcode.aspx?id=<%= channel.ID%>">下载二维码</a>
                                    <a class="btnGrayS vm doaddit"href="
ChannelEdit.aspx?id=<%= channel.ID%>">编辑</a>
                                    <a href="javascript:void(0)" data="<%=
channel. ID%>" class="dodelit deleteChannel">删除</a>
                                </p>
                            </td>
                        </tr>
                    <% } %>
                </tbody>
            </table>
        </form>
    </div>
    <script>
        $(function(){
            $(".deleteChannel").click(function(){
                if (confirm("确定删除吗? ")) {
                    var id = $(this).attr("data");
                    $.ajax({
                        url: "ChannelDelete.aspx?id=" + id,
                        type: "get",
                        success: function (repText) {
                            if (repText && repText == "True") {
                                window.location.reload();
                            } else {
                                alert(repText.ErrorMsg);
                            }
                        },
                        complete: function (xhr, ts) {
                            xhr = null;
                        }
                    });
                }
            });
        });
    </script>
</body>
</html>
```

删除操作对应的 ChannelDelete.aspx 的代码如下：

```
public partial class ChannelDelete : System.Web.UI.Page
{
    /// <summary>
    /// 删除渠道
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        //获取渠道 ID
        int id;
        if (int.TryParse(Request.QueryString["id"], out id))
        {
            //删除渠道并返回删除结果
            bool result = new ChannelBll().DeleteEntityById(id);
            Response.Write(result.ToString());
            Response.End();
        }
    }
}
```

下载二维码操作对应的 QRcode.aspx 的代码如下：

```
public partial class QRcode : System.Web.UI.Page
{
    /// <summary>
    /// 下载渠道二维码
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        //获取渠道 ID
        int id;
        if (int.TryParse(Request.QueryString["id"], out id))
        {
            var entity = new ChannelBll().GetEntityById(id);
            //将数据库中 Base64String 格式图片转换为 Image 格式，返回给浏览器
            string base64Image = File.ReadAllText(System.Web.HttpContext.
Current.Server.MapPath("~/App_Data/") + entity.Qrcode);
            byte[] arr = Convert.FromBase64String(base64Image);
            MemoryStream ms = new MemoryStream(arr);
```

```
            Response.ContentType = "image/jpeg";
            ms.WriteTo(Response.OutputStream);
            Response.End();
        }
    }
}
```

4. 推广渠道二维码扫描记录列表

Sample_5 示例程序只是将所有包含微信用户信息扫描记录进行了列表显示。在实际项目中还需要按第 7 章中提供的方法绘制每个推广渠道按时间段统计的扫描次数曲线图，并可在一个图表中对比显示。还需要对每个推广渠道的扫描记录进行计算，给出统计数据与统计对比。

推广渠道二维码扫描记录列表由 Web 窗体 ChannelScanList 实现。

```
public partial class ChannelScanList : System.Web.UI.Page
{
    /// <summary>
    /// 渠道扫描记录列表
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected void Page_Load(object sender, EventArgs e)
    {
        //获取渠道 ID
        int id;
        if (int.TryParse(Request.QueryString["id"], out id))
        {
            //获取渠道扫描记录列表数据
            ViewState["ChannelScanDisplayList"] = new ChannelScanBll().
GetChannelScanList(id);
        }
    }
}
```

ChannelScanList 的页面代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ChannelScanList.
aspx.cs" Inherits="Sample_5.ChannelScanList" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>扫描记录列表</title>
    </head>
    <!-- #Bootstrap -->
    <link href="Css/bootstrap.min.css" rel="stylesheet" />
    <script src="Scripts/bootstrap.min.js"></script>
    <!-- #Jquery -->
    <script src="Scripts/jquery-1.9.1.min.js"></script>
    <body>
        <div class="container">
            <form id="Form1" class="form-horizontal" type="post" runat="server">
                <h2 class="form-signin-heading">扫描记录列表</h2>
                <table id="listTable" class="table table-bordered table-hover
dataTable" style="width: auto">
                    <thead>
                        <tr>
                            <th style="border-color: #ddd; color: Black">ID
                            </th>
                            <th style="border-color: #ddd; color: Black">扫描类型
                            </th>
                            <th style="border-color: #ddd; color: Black">扫描时间
                            </th>
                           <th style="border-color: #ddd; color: Black">所属渠道 ID
                            </th>
                            <th style="border-color: #ddd; color: Black">微信用户
OpenId
                            </th>
                            <th style="border-color: #ddd; color: Black">头像
                            </th>
                            <th style="border-color: #ddd; color: Black">昵称
                            </th>
                            <th style="border-color: #ddd; color: Black">性别
                            </th>
                            <th style="border-color: #ddd; color: Black">国家
                            </th>
                            <th style="border-color: #ddd; color: Black">省
                            </th>
                            <th style="border-color: #ddd; color: Black">市
                            </th>
                            <th style="border-color: #ddd; color: Black">关注时间
                            </th>
                        </tr>
                    </thead>
```

```
                <tbody>
                    <% foreach (Sample_5.ViewEntity.ChannelScanDisplayEntity
channelScanDisplay in (ViewState["ChannelScanDisplayList"] as List<Sample_5.
ViewEntity.ChannelScanDisplayEntity>))
                                { %>
                            <tr>
                                <td><%= channelScanDisplay.ScanEntity.ID%>
                                </td>
                                <td><%= channelScanDisplay.ScanEntity.ScanType.
ToString()%>
                                </td>
                                <td><%= channelScanDisplay.ScanEntity.ScanTime.
ToString()%>
                                </td>
                               <td><%= channelScanDisplay.ScanEntity.ChannelId.
ToString()%>
                                </td>
                                <td><%= channelScanDisplay.ScanEntity.OpenId%>
                                </td>
                               <td><img width="50px" src="<%= channelScanDisplay.
UserInfoEntity.HeadImgUrl%>" />
                                </td>
                            <td><%= channelScanDisplay.UserInfoEntity.NickName%>
                                </td>
                                <td><%= channelScanDisplay.UserInfoEntity.Sex.
ToString()%>
                                </td>
                            <td><%= channelScanDisplay.UserInfoEntity.Country%>
                                </td>
                            <td><%= channelScanDisplay.UserInfoEntity.Province%>
                                </td>
                               <td><%= channelScanDisplay.UserInfoEntity.City%>
                                </td>
                                <td><%= channelScanDisplay.UserInfoEntity.
SubscribeTime.ToShortDateString()%>
                                </td>
                            </tr>
                        <% } %>
                    </tbody>
                </table>
            </form>
        </div>
    </body>
    </html>
```

8.4 部署及测试体验

在 Sample_5 项目本地调试通过后，将项目上传到 AppHarbor 中进行部署，使微信服务器能访问到我们的程序，进行实际测试。AppHarbor 部署的详细步骤见第 4 章。

在部署完毕后，首先打开推广渠道类型列表页面 ChannelTypeList.aspx，添加一个渠道类型“线上渠道”，如图 8-2 所示。

渠道类型列表

添加 渠道列表

ID	渠道类型名称	操作
1	线上渠道	编辑 删除

图 8-2

然后打开推广渠道列表页面 ChannelList.aspx，添加一个渠道“豆瓣推广”，如图 8-3 所示。

渠道列表

添加

ID	场景值ID	渠道名称	所属渠道类型ID	操作
1	1	豆瓣推广	1	查看扫描记录 下载二维码 编辑 删除

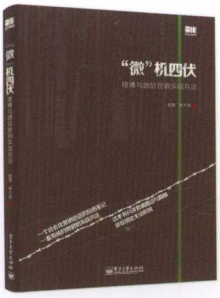
图 8-3

单击推广渠道列表页面的“下载二维码”按钮，将在新窗口中打开“豆瓣推广”渠道的二维码。使用微信扫描该二维码后，单击推广渠道列表页面的“查看扫描记录”按钮，在二维码扫描记录列表页面可以看到刚才的扫描记录已经显示出来，如图 8-4 所示。

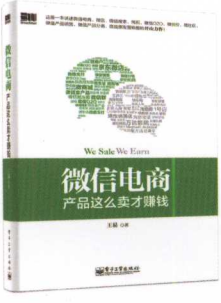
扫描记录列表

ID	扫描类型	扫描时间	所属渠道ID	微信用户Openid	头像	昵称	性别	国家
1	Scan	2014/5/25 17:29:46	1	ochtHuNLNpQOm2EPnWocvqmH0JOw		刘捷	Male	中国
2	Scan	2014/5/25 17:31:35	1	ochtHuNLNpQOm2EPnWocvqmH0JOw		刘捷	Male	中国

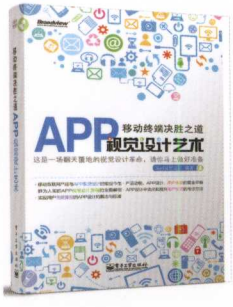
图 8-4



《“微”机四伏——微博与微信营销实战兵法》
ISBN 978-7-121-23870-3



《微信电商，产品这么卖才赚钱》
ISBN 978-7-121-23455-2



《移动终端决胜之道
——APP视觉设计艺术》
ISBN 978-7-121-24500-8



《C#编程兵书》
ISBN 978-7-121-20740-2

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

微信公众平台
企业应用开发实战



如果你符合下列情况之一，那么，这正是你需要的图书！

- 准备使用或正在使用微信公众平台扩大业务的管理者
- 微信公众账号运营、推广、管理等的从业者
- 对微信公众平台企业应用有需求的IT或市场人员
- 准备投身于或正在从事微信公众平台应用开发的工程师
- 从事微信公众平台企业级开发的专业开发团队

本书内容预览

- 微信公众平台基础知识，微信公众账号的类型及特点、微信公众账号的注册与认证流程、功能插件介绍，等等。
- 微信公众平台开发所需的准备工作，以及如何将应用接入微信公众平台。
- 微信公众平台已开放的API，应用方法及技巧详解。
- 实际项目剖析，源代码全面解析，成熟开发框架大放送。
- 微预约、微统计、微渠道，全真微信公众平台第三方应用开发案例解析。

上架建议：程序设计 > 移动开发

博文视点Broadvview

新浪微博 weibo.com
@博文视点Broadview

策划编辑：牛　勇
责任编辑：徐津平
封面设计：侯士卿

ISBN 978-7-121-25013-2

定价：49.80元